THE MICROSOFT JOURNAL FOR DEVELOPERS

DECEMBER 2013 VOL 28 NO 12

msdn magazine

EF

Exploring Entity Framework 6.....20

Create mobile apps with HTML5, JavaScript and Visual Studio

Your next great app starts here.

From interactive Desktop applications, to immersive Web and Mobile solutions, development tools built to meet your needs today and ensure your continued success tomorrow. Download your free 30-day trial today and experience the DevExpress Difference.

devexpress.com/try

DevExpress™

THE MICROSOFT JOURNAL FOR DEVELOPERS

DECEMBER 2013 VOL 28 NO 12

msdn magazine

Exploring Entity Framework 6.....20

Entity Framework 6: The Ninja Edition
Julie Lerman.....20

CORS Support in ASP.NET Web API 2
Brock Allen.....30

Cross-Browser, Coded UI Testing with Visual Studio 2013
Damian Zapart.....40

An Introduction to Model-Based Testing and Spec Explorer
Yiming Cao and Sergio Mera.....48

Freedom of Information Act Data at Your Fingertips
Vishwas Lele.....54

Rendering PDF Content in Windows Store Apps
Sridhar Poduri.....62

COLUMNS

WINDOWS AZURE INSIDER
Meet the Demands of Modern Gaming with Windows Azure
Bruno Terkaly and Ricardo Villalobos, page 6

TEST RUN
Radial Basis Function Network Training
James McCaffrey, page 12

THE WORKING PROGRAMMER
Getting Started with Oak: A Different Approach
Ted Neward, page 66

MODERN APPS
Everything You Need to Know About the WinJS ListView Control
Rachel Appel, page 70

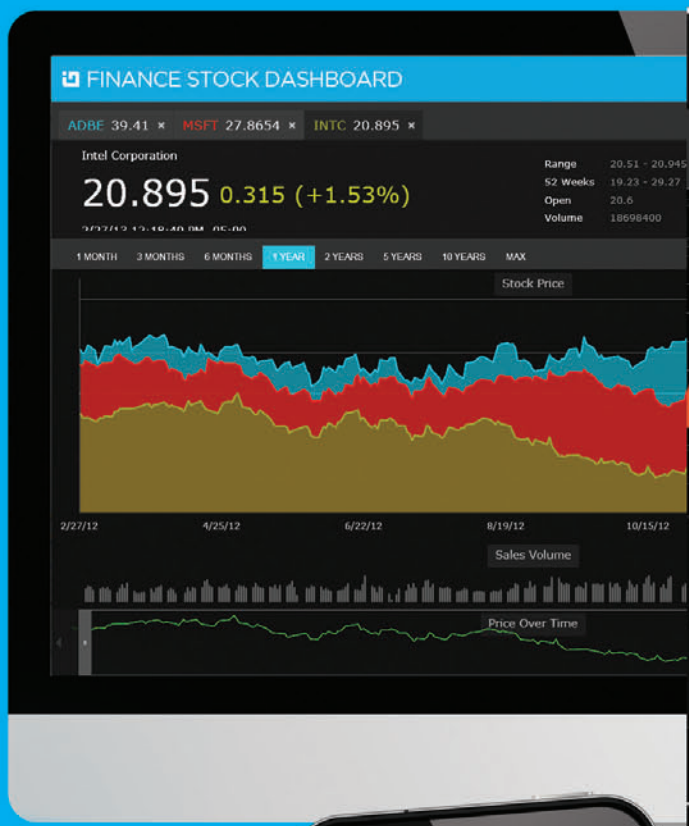
DIRECTX FACTOR
Character Outline Geometries Gone Wild
Charles Petzold, page 74

DON'T GET ME STARTED
Original Sin?
David Platt, page 80

Microsoft

Desktop

Deliver high performance, scalable and stylable touch-enabled enterprise applications in the platform of your choice.



Native Mobile

Develop rich, device-specific user experience for iOS, Android, and Windows Phone, as well as mobile cross-platform apps with Mono-Touch.

Download Your Free Trial

infragistics.com/enterprise-READY

Dr.Dobb's JOLT 2013 BEST PROGRAMMER LIBRARIES

Cross-Device

Build standards-based, touch-enabled HTML5 & jQuery experiences for desktop, tablet, and mobile delivery, including multi-device targeting with frameworks such as PhoneGap and MVC.

UX Prototyping

Explore design ideas through rapid, user-centered wireframing, prototyping, and evaluation before investing a single line of code.

INFRAGISTICS
DESIGN / DEVELOP / EXPERIENCE

Infragistics Sales US 800 231 8588 • Europe +44 (0) 800 298 9055 • India +91 80 4151 8042 • APAC (+61) 3 9982 4545

msdn magazine

DECEMBER 2013 VOLUME 28 NUMBER 12

MSDN Magazine (ISSN 1528-4859) is published monthly by 1105 Media, Inc., 9201 Oakdale Avenue, Ste. 101, Chatsworth, CA 91311. Periodicals postage paid at Chatsworth, CA 91311-9998, and at additional mailing offices. Annual subscription rates payable in US funds are: U.S. $35.00, International $60.00. Annual digital subscription rates payable in U.S. funds are: U.S. $25.00, International $25.00. Single copies/back issues: U.S. $10, all others $12. Send orders with payment to: *MSDN Magazine*, P.O. Box 3167, Carol Stream, IL 60132, email MSDNmag@1105service.com or call (847) 763-9560. **POSTMASTER:** Send address changes to *MSDN Magazine*, P.O. Box 2166, Skokie, IL 60076. Canada Publications Mail Agreement No: 40612608. Return Undeliverable Canadian Addresses to Circulation Dept. or XPO Returns: P.O. Box 201, Richmond Hill, ON L4B 4R5, Canada.

LEADTOOLS®

The world's leading Imaging SDK
RUNS ANYWHERE

Document	Medical	Multimedia	Imaging
OCR, Barcode & Forms Recognition	*DICOM*	*Playback, Capture & Conversion*	*Viewers*
PDF Read, Write & Edit	*PACS*	*MPEG-2 Transport Stream*	*Image processing*
Cleanup and Preprocessing	*Medical Workstation*	*DVR*	*150+ Formats*

C++ C# JavaScript VB Objective-C Java

.NET Windows API WinRT Linux iOS OS X Android HTML5

DOWNLOAD OUR 60 DAY EVALUATION
WWW.LEADTOOLS.COM

LEAD TECHNOLOGIES INCORPORATED

SALES@LEADTOOLS.COM
800.637.1840

MICHAEL DESMOND

In Praise of Entity Framework 6

There's a reason that Julie Lerman's Data Points column is consistently among the most widely read in *MSDN Magazine*. In addition to being a whip-smart programmer with a keen sense for what developers need, Lerman covers that most core and enduring aspect of programming—managing data. Development platforms come and go, but the challenges of wrangling data always remain.

Which is why this month's issue of *MSDN Magazine* leads off with a feature on Entity Framework 6, the latest version of the Microsoft-founded, object-relational mapping (ORM) framework. Lerman took a break from her usual column-writing schedule to pen the feature, titled "Entity Framework 6: The Ninja Edition" (p. 20). As Lerman points out in the article, Entity Framework 6 represents a major step forward from Entity Framework 5, boasting improved performance, reduced complexity and features that allow for more advanced development scenarios.

> "Some bigger features are getting a lot of airplay. But there are so many other smaller ones that may not be used by as many developers but will have a big impact for those who do use them."

Notably, the latest version of Entity Framework was developed under an open source development model and has been decoupled from the long release cycles of the Microsoft .NET Framework. As Lerman reports, the Entity Framework APIs have been extracted from the .NET Framework, allowing timely updates and enabling compatibility with both .NET Framework 4 and 4.5. Microsoft also disentangled the Entity Framework 6 Designer from Visual Studio, casting it as a Visual Studio extension that the Entity Framework team is able to update independently of Visual Studio updates. You can find the project hosted on CodePlex at entityframework.codeplex.com.

"I think one of the most important things about EF6 [Entity Framework 6] is the fact that it's now completely open source," Lerman says, noting the contributions of developers like Unai Zorrilla, whose AddRange and RemoveRange APIs streamline the addition and removal of multiple entities in Entity Framework 6.

Many developers are familiar with the major features of Entity Framework 6, such as support for asynchronous querying and saving, and support for stored procedure mapping in Code First models. But Lerman says a host of smaller, less-publicized changes can really impact development. Two examples she points out are the ability to combine calls in a shared DbTransaction and the ability to reuse opened database connections.

"Some bigger features are getting a lot of airplay. But there are so many other smaller ones that may not be used by as many developers but will have a big impact for those who do use them," Lerman says.

While Entity Framework 6 has delivered many improvements, Lerman says she's looking forward to a few capabilities that didn't make it into the latest version of the framework. Key among them is better tooling for reverse engineering to Code First classes and DbContext, which Lerman says will make it easier to start with an existing database and create constrained models used with Domain-Driven Design (DDD) architectural patterns.

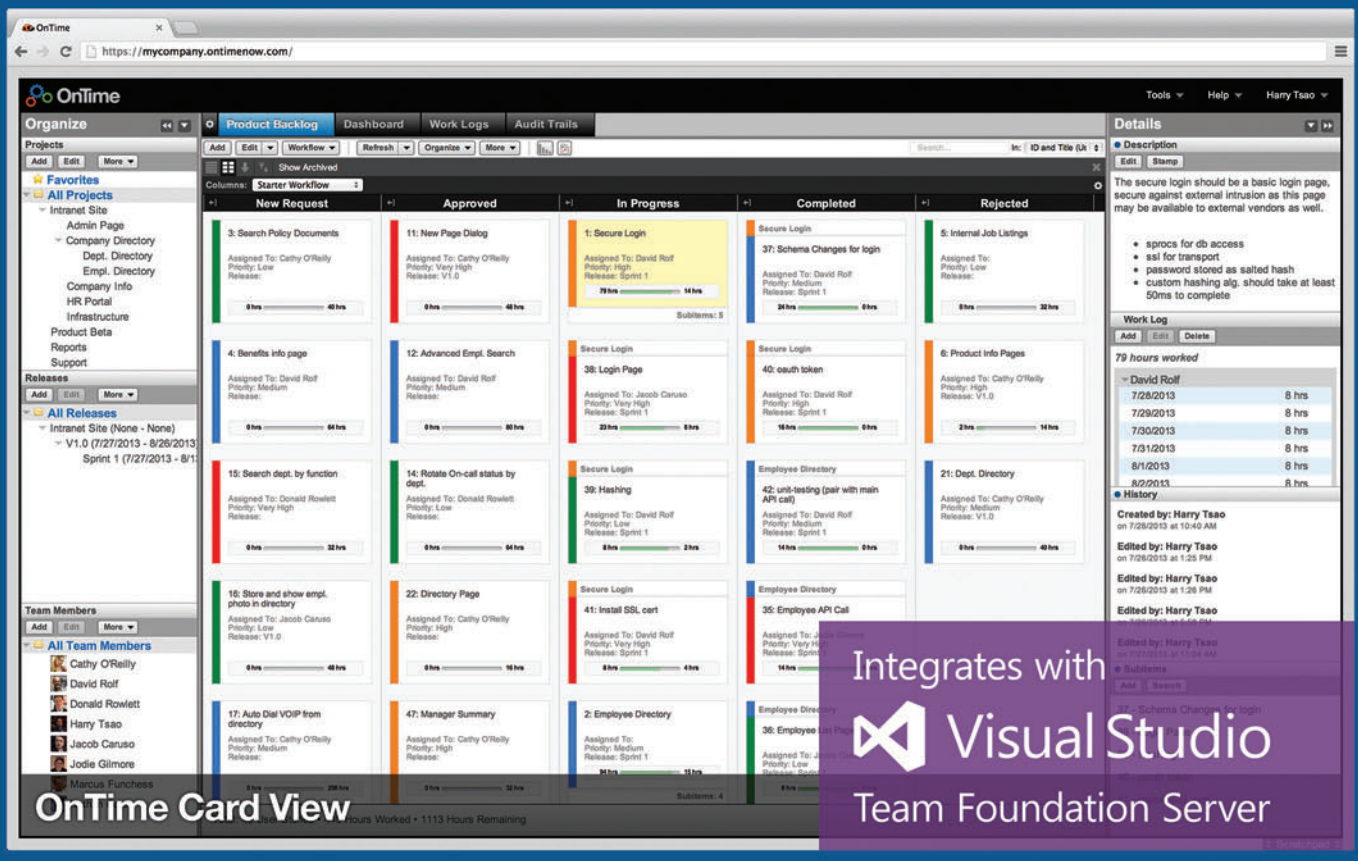
I asked Lerman to tell me why a developer *wouldn't* want to consider moving up to the latest version of Entity Framework. She was direct.

"As long as the Entity Framework provider you're using has been updated to be compatible for EF6, I can't think of one. Even if you're using .NET 4, EF4, EntityObject and ObjectContext, other than some namespace changes, you really don't change any code. You can move to EF6 and benefit from the faster view generation, and even take advantage of some of the new features like reusing open DbConnections," she explains. However, she cautions that developers should do some testing before jumping into that scenario.

There's a whole lot to like in Entity Framework 6, and Lerman's feature on the new ORM tool is well worth a read for those hoping to take advantage of it.

Visit us at msdn.microsoft.com/magazine. Questions, comments or suggestions for *MSDN Magazine*? Send them to the editor: mmeditor@microsoft.com.

the #1 selling scrum software



OnTime Scrum agile project management software

Want **three extra months of dev time** a year? Operate like our OnTime Scrum customers who have been able to ship their software 24% faster!

So how are they doing it? Our fast, customizable interface allows users save time and zip from one task to another on one screen—no clicking around. From there **Card View** creates a visually appealing, intuitive system for managing user stories that integrates seamlessly into Visual Studio and TFS source control—perfect for any **Microsoft development environment**.

Is it any wonder that OnTime Scrum is the **#1 selling Scrum software**?

just **$7** per user per month

Get 10% off your OnTime subscription with this link:

OnTimeNow.com/MSDN

Plus: learn about our solutions for bug tracking, help desk & customer management, and project wikis

axosoft

800.653.0024 • www.ontimenow.com • www.axosoft.com • @axosoft

BRUNO TERKALY AND
RICARDO VILLALOBOS

Meet the Demands of Modern Gaming with Windows Azure

Online, mobile and social games have taken the world by storm, with staggering numbers relating to concurrent players, time spent playing and downloaded applications. Just recently, Facebook shared that one third of its global users (260 million out of 750 million) actively play games on its desktop and mobile sites. This is just one indicator of the particular challenges game developers and publishing companies have to face—challenges that include users playing the same game on multiple platforms and devices, players expecting to receive instant notifications when the status of their time-based game has changed, games going from startup to viral in a matter of days, and trying to reach audiences in multiple locations around the world.

Thankfully, the public cloud offers a number of alternatives to deal with these situations, allowing you to concentrate on developing your game applications, and not on how to provision the infrastructure to support authentication, computing, data or media requirements. In this article, we'll explore how to solve these and other common scenarios using the latest Windows Azure services and components.

Figure 1 shows a typical architecture for supporting multiplayer games for mobile clients in the cloud, either for turn-based or real-time scenarios. The different components are assigned to the most common tasks required to build a gaming back end, introducing the concept of an orchestrator or proxy, which acts as the gateway or traffic controller for all the client interactions with multiple services.

In the following paragraphs, we'll take a closer look at each of these components.

Multiplayer Game Servers (Infrastructure as a Service Virtual Machines)

Multiplayer game servers are usually based on open source or licensed frameworks, acting as the authoritative source of events for the clients connected to them and providing information about other players who have joined the same server via low-latency calls. These frameworks require stateful interactions with clients, as well as access to local storage, making virtual machines (VMs) in Windows Azure the perfect choice for this component. Some examples of these game server frameworks are pomelo (bit.ly/1i9heBe), Marauroa (bit.ly/am9MOi) and Photon Server (exitgames.com).

To deploy any of these frameworks to VMs, you can create instances from the Windows Azure image gallery in the management portal (bit.ly/197eXED), or directly create custom VM images as VHD (Hyper-V) files (bit.ly/PQso1a).

Multiplayer game servers are usually based on open source or licensed frameworks, acting as the authoritative source of events for the clients connected to them.

Keep in mind that implementing a game server can turn into a complex process requiring you to come up with solutions for network-lag compensation and for properly using communication protocols (such as TCP or UDP) to optimize communication with the clients.

Gaming Orchestrator or Proxy (Platform as a Service Cloud Services)

The heart of the solution is the gaming orchestrator or proxy, which provides multiple services, including the following:

1. Authentication: Validating credentials sent by the clients before players can connect to the game servers.
2. Matchmaking: Assigning players with similar preferences to the same game server. The decision could also be driven by location, to minimize latency.
3. Data proxy: Serving in-game requests from the game servers or the clients; interacting with the external storage; and sending back data such as historical scores, profile information, preferences, or credit balance.
4. Provisioning: Increasing or decreasing the number of VMs, using a scale-out approach based on the number of connected players.

This article describes the Windows Azure Cache Service, which is in preview. Information is subject to change.

telerik

deliver more than expected

Telerik DevCraft Q3

RELEASE: OCTOBER 2013

- Telerik's full stack of .NET controls and tools for the professional developer
- 350+ UI controls and reporting for all Microsoft platforms
- Productivity tools for faster coding, debugging and profiling

See what's new in Q3 2013 Release & download your 30 day free trial at

www.telerik.com

5. Notifications: Interacting with the notification service to inform players of the status of the game when they're not online. This should support multiple platforms (Windows, Android, iOS) and device types.
6. Delegation: Orchestrating the interaction with external services, including but not limited to sending e-mail messages, processing payments, and logging game information.

Windows Azure Cloud Services is the perfect candidate for the gaming orchestrator, which must be able to handle stateless requests, as well as easily scale out based on the number of client and game server requests. Moreover, thanks to the benefits offered by virtual networks in Windows Azure, Cloud Services can directly communicate with the game servers hosted in VMs, adding an extra layer of security to the architecture by not having to use external endpoints. Cloud Services can be created and deployed from different IDEs, including Visual Studio for .NET development or Eclipse for Java. More information about creating Cloud Services can be found at bit.ly/19MYq5A. If you want to connect VMs and Cloud Services using virtual networks, the tutorial at bit.ly/GYcG5t can help.

The responsibilities just discussed can be handled by a single cluster of Web or worker roles, or split into many Cloud Services, depending on the number of concurrent users and complexity of the online game. One of the benefits Windows Azure provides is multiple deployment models, including Infrastructure as a Service (IaaS) or Platform as a Service (PaaS). The key decision factor is the number of software layers delegated to the cloud vendor—in this case, Microsoft. More information about choosing the right cloud deployment and execution model can be found at bit.ly/153kRXM.

In modern gaming, it's important to support validation from multiple identity providers, keeping in mind that users have different preferences when it comes to online security.

Now we'll explore how the orchestrator can perform these responsibilities using other components in the Windows Azure platform.

Authentication (Windows Azure Active Directory Access Control Service)

The first step a mobile client takes to access a multiplayer platform is trying to authenticate against the server using a set of credentials. In modern gaming, it's important to support validation from multiple identity providers, keeping in mind that users have different preferences when it comes to online security. Some might feel more comfortable using social network credentials, such as Facebook, Yahoo! or Twitter. Others might prefer authentication provided by the game itself, based on a framework such as Active Directory.

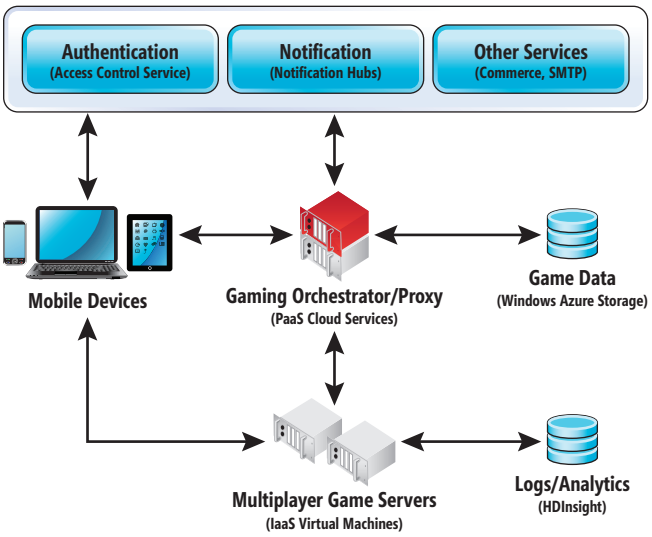


Figure 1 **Software Architecture for Real-Time, Multiplayer Games for Mobile Clients**

The Windows Azure access control service (ACS) offers a simple way to perform this authentication. It supports integration with Windows Identity Foundation; provides out-of-the-box compatibility with Windows Live ID (a Microsoft account), Google, Yahoo! and Facebook; enables scenarios based on OAuth 2.0, WS-Trust and WS-Federation protocols; and recognizes JSON Web Token (JWT), SAML 1.1, SAML 2.0 and Simple Web Token (SWT) token formats.

In this case, the game client obtains a security token issued by ACS in order to log on to the game server via the gaming proxy. This token contains a set of claims about the user's identity. ACS does not issue a token unless the user first proves a valid identity by presenting a security token from another trusted issuer or identity provider that has authenticated the user. This process is illustrated in **Figure 2**.

Matchmaking

In multiplayer video games, the process of assigning gamers to the appropriate game server based on their preferences or location is called matchmaking. Once the game client has been authenticated, the gaming proxy will return a list of game servers matching the player's preferences, based on his IP address, along with additional security credentials in case they're needed by the game server itself. The logic behind the assignment is usually based on a list of recommended servers in each datacenter that's stored in some type of caching mechanism. The reason caching is recommended for this is performance: avoiding round-trips to primary storage improves the gamer experience significantly. Windows Azure offers a new cache service (currently in preview) that can be accessed across multiple services and clients. This new service is a distributed, in-memory, scalable solution that enables you to build highly responsive applications by providing super-fast access to data. It's extremely easy to access from .NET applications using the corresponding SDK. Detailed instructions can be found at bit.ly/15lItBt. The matchmaking process will return the appropriate IP address of the game server to which the game client should connect.

Building Blocks for Global Data Quality Success

A strong foundation for enterprise data starts with Melissa Data.

Our powerful, scalable data cleansing and integration tools help you profile, cleanse, consolidate, and enrich your contact data. Gain a better understanding of your customer, vendor and supplier data; improve deliverability; increase cost savings; and enhance your operational efficiencies with Melissa Data.

Advanced Functionalities:

- Verify, correct, and enhance addresses for 240+ countries
- Add lat/long coordinates to addresses all over the world
- Match and consolidate data to create the golden record
- Append missing contact data like phone numbers and email addresses
- Integrate into many technologies with multiplatform capabilities

www.MelissaData.com
or call 1-800-MELISSA (635-4772)

MELISSA DATA®
Your Partner in Data Quality

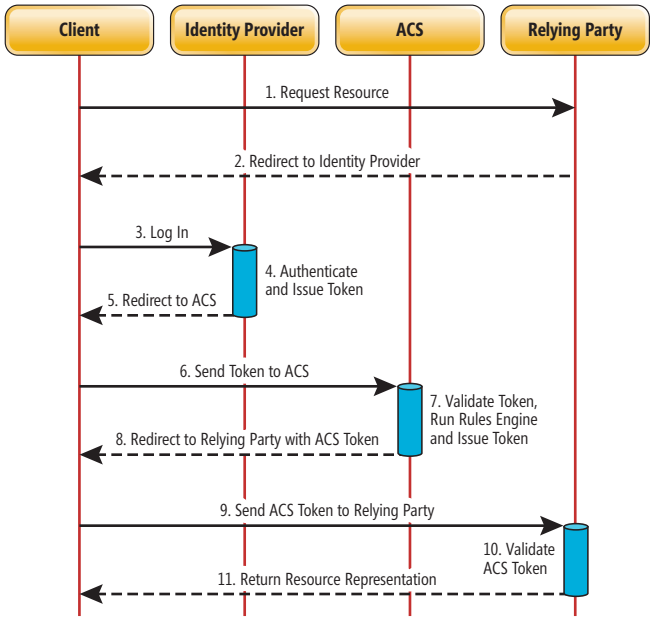


Figure 2 **The Authentication Process Using Windows Azure Access Control Service**

When a game session has been finalized (in the form of a match, combat, or simply by the gamer logging out of the game), players can stay on the same server or be redirected back to the match-making process, in order to locate a different server in case their preferences have changed.

Data Proxy (Windows Azure Storage)

For players already connected to the appropriate game server based on their preferences, requests about game configuration, store balance or any other user-related data are routed through the game orchestrator/proxy, which usually exposes an API with the most common operations against a data repository. There are two main decisions to make for this: the storage mechanism for the user and game information, and the framework for exposing services that can be accessed by the game servers.

> Windows Azure also offers a turnkey service that combines the power of a relational database in the cloud with a robust and flexible REST API.

Windows Azure offers multiple options for storing information, which we've described in previous articles. Depending on the number and complexity of the queries, the structure of the data, and the level of customization required, you can choose from traditional relational repositories such as Windows Azure SQL Database or NoSQL approaches such as Windows Azure table storage (bit.ly/YrYcQP).

Regardless of the repository selection, services in front of this information should be exposed in a RESTful manner, using frameworks such as Windows Communication Foundation (WCF) or the recently released ASP.NET Web API (asp.net/web-api). These frameworks can be deployed to cloud services or VMs.

Windows Azure also offers a turnkey service that combines the power of a relational database in the cloud with a robust and flexible REST API. It's called Windows Azure Mobile Services, and it's an easy way to accelerate the development of this piece of the gaming architecture, with easy-to-follow wizards and auto-scaling capabilities. More information about this service can be found in our November 2012 column, "Windows Azure Mobile Services: A Robust Back End for Your Device Applications" (msdn.microsoft.com/magazine/jj721590), or on the official Windows Azure page (bit.ly/188Llcg).

Provisioning

The game orchestrator/proxy can also act as the provisioning or auto-scaling engine for adding or removing game servers as they're needed. However, note that Windows Azure now offers auto-scaling capabilities for all of the different deployment models, including VMs, Cloud Services, and Web sites.

Notifications

Gamers using mobile devices rely on receiving notifications when they're offline, particularly for turn-based or time-based games that require keeping virtual properties or items up-to-date (a concept that was made extremely common by games such as FarmVille). You face two main problems when sending notifications to mobile gamers: having to reach out to players using different platforms on their mobile devices, and building and maintaining an infrastructure capable of reaching millions of users. Thankfully, Windows Azure offers a service called Notification Hubs, which supplies a common API to send push notifications to a variety of mobile platforms, including Windows Store, Windows Phone, iOS and Android. At the same time, the push notifications are sent to millions of users within minutes, not hours. **Figure 3** shows a code snippet in C# that sends a simple notification to both Windows Store and iOS applications using multiple categories.

Figure 3 **Sending Notifications to Windows Store and iOS Applications Using Windows Azure Notification Hubs**

```
private static async void SendNotificationAsync()
{
  NotificationHubClient hub = NotificationHubClient.
CreateClientFromConnectionString(
  "<connection string with full access>", "<hub name>");
  var categories = new string[] { "World", "Politics", "Business",
    "Technology", "Science", "Sports"};
  foreach (var category in categories) {
    var toast = @"&lt;toast&gt;&lt;visual&gt;&lt;binding template ="
      "ToastText02""&gt;&lt;text id=""1""&gt;" + "Breaking " + category +
        " News!" + "&lt;/text&gt;&lt;/binding&gt;&lt;/visual&gt;&lt;/toast&gt;";
    await hub.SendWindowsNativeNotificationAsync(toast, category);
    var alert = "{\"aps\":{\"alert\":\"Breaking "+ category +" News!\"}}";
    await hub.SendAppleNativeNotificationAsync(alert, category);
  }
}
```

Delegation of Responsibilities to Other Services (SMTP, Commerce)

The game orchestrator/proxy should also act as the gateway for accessing other services, such as engines for sending massive e-mail messages via SMTP, or processing game images in real time. The Windows Azure Store lets you quickly discover, purchase and provision applications from other Microsoft partners. These services can be combined with Windows Azure components to build complex games or add features that are not natively supported by the cloud platform. More information about the Windows Azure Store and a catalog of developer services can be found at bit.ly/1carBrd.

Game Analytics and Big Data

Collecting, analyzing and reporting data gathered from game servers is crucial to finding bottlenecks, improving in-game performance and, for monetization and advertising purposes, determining areas where gamers spend their time. Windows Azure lets you use a MapReduce approach, called HDInsight, as a service, which allows the simple, straightforward installation of Hadoop clusters. Using Hive, or even familiar tools such as Microsoft Office, you can create rich reports and charts. More information about this component can be found in our September 2013 column, "Hadoop and HDInsight: Big Data in Windows Azure" (msdn.microsoft.com/magazine/dn385705).

Wrapping Up

We've only scratched the surface on how to create rich and scalable solutions for the mobile game industry by combining multiple components of the Windows Azure platform, to meet the demands and requirements created by social networks and the massive number of players that are attracted to these applications. The public cloud offers a flexible pay-as-you-go model, which enables companies of all sizes to compete in this space. ■

Bruno Terkaly *is a developer evangelist for Microsoft. His depth of knowledge comes from years of experience in the field, writing code using a multitude of platforms, languages, frameworks, SDKs, libraries and APIs. He spends time writing code, blogging and giving live presentations on building cloud-based applications, specifically using the Windows Azure platform. You can read his blog at blogs.msdn.com/b/brunoterkaly.*

Ricardo Villalobos *is a seasoned software architect with more than 15 years of experience designing and creating applications for companies in multiple industries. Holding different technical certifications, as well as a master's degree in business administration from the University of Dallas, he works as a cloud architect in the DPE Globally Engaged Partners team for Microsoft, helping companies worldwide to implement solutions in Windows Azure. You can read his blog at blog.ricardovillalobos.com.*

Terkaly and Villalobos jointly present at large industry conferences. They encourage readers of Windows Azure Insider to contact them for availability. Terkaly can be reached at bterkaly@microsoft.com and Villalobos can be reached at Ricardo.Villalobos@microsoft.com.

Thanks to the following technical expert for reviewing this article: *Kevin Ashley (Microsoft)*

JAMES MCCAFFREY

Radial Basis Function Network Training

A radial basis function (RBF) network is a software system that can classify data and make predictions. RBF networks have some superficial similarities to neural networks, but are actually quite different. An RBF network accepts one or more numeric inputs and generates one or more numeric outputs. The output values are determined by the input values, plus a set of parameters called the RBF centroids, a set called the RBF widths, a set called the RBF weights and a set called the RBF biases.

For simplicity of terminology, the combination of RBF weights and bias values are sometimes collectively referred to as the weights. The context in which the term weights is used usually makes the meaning clear. For more information, see my article, "Radial Basis Function Networks for Programmers" in the October issue of *MSDN Magazine* (msdn.microsoft.com/magazine/dn451445).

When using an RBF network for classification and prediction, the challenge is to find a set of values for the centroids, widths, weights and biases so that computed outputs best match a set of known outputs. This is called training the RBF network. Although researchers have studied RBF networks since their introduction in 1988, there's not much practical guidance that explains how to implement RBF network training. This article will present and describe a complete demo RBF network.

Take a look at the demo program in **Figure 1**. The program creates an RBF model that predicts the species of an iris flower ("setosa," "versicolor" or "virginica") from four numeric values for the flower's sepal length and width, and petal length and width. The demo program's data source consists of 30 items that are a subset of a well-known 150-item benchmark set called Fisher's Iris data. The 30 data items have been preprocessed. The four numeric x-values have been normalized so that values less than zero mean shorter-than-average length or width, and values greater than zero mean longer-than-average length or width. The y-value for species has been encoded as (0,0,1), (0,1,0), or (1,0,0) for setosa, versicolor, and virginica, respectively.

The demo splits the 30-item data set into a 24-item set to be used for training. There's also a holdout six-item set for testing/evaluation of the resulting RBF model. It instantiates an RBF network with four input nodes (one for each input data value), five hidden processing nodes and three output nodes (one for each output data value). Determining the best number of hidden nodes is mostly a matter of trial and error. The choice of five in the demo was arbitrary.

Code download available at archive.msdn.microsoft.com/mag201312TestRun.

In **Figure 1** you can see that training an RBF network consists of three phases. The first phase determines the centroids. You can think of centroids as representative x-values selected from the training data. An RBF network requires one centroid for every hidden node, so the demo needs five centroids. The training algorithm selects the x-values from training data items [9], [19], [21], [20] and [4]. In other words, the first centroid is (-0.362, -2.019, 0.074, 0.112).

The second phase of training determines widths. You can think of widths as values that describe the distance between the centroids. An RBF network requires one width for every hidden node. The demo computes a single common width with the value 3.3318 for all five hidden nodes, rather than computing five separate widths.

The third phase of training determines the RBF weights and bias values. You can think of weights and bias values as numeric constants. If an RBF network has NI number of input nodes, NH number of hidden nodes, and NO number of output nodes, then the network requires (NH * NO) weight values and NO bias values. So, because the demo RBF network has a 4-5-3 architecture, it needs 5 * 3 = 15 weights plus three biases, for a total of 18 weights and bias values. The demo program uses particle swarm optimization (PSO) to determine the 18 weights and biases.

After you've trained the demo RBF network using the 24-item training data set, you feed the six-item test data set into the network. In this example, the RBF network correctly predicts the species of five out of the six test items, for a classification accuracy of 0.8333.

This article assumes you have advanced programming skills with C# and a basic familiarity with the radial basis function network input-process-output mechanism. I discussed that mechanism in my October column. The source code for the demo program is too long to present in its entirety in this article, but the complete code download is available at archive.msdn.microsoft.com/mag201312TestRun.

Overall Program Structure

To create the demo program, I launched Visual Studio 2012 and created a C# console application named RadialNetworkTrain. The demo has no significant .NET dependencies so any version of Visual Studio should work. After the template code loaded, in the Solution Explorer window I renamed file Program.cs to the more descriptive RadialTrainProgram.cs and Visual Studio automatically renamed associated class Program. At the top of the source code, I deleted all unnecessary references to .NET namespaces, leaving just the reference to the System namespace.

The overall program structure, with some WriteLine statements removed and a few minor edits, is presented in **Figure 2**. In addition

to the program class that houses the Main method, the demo has a RadialNetwork class that encapsulates RBF network creation and training, a Particle class that defines a particle object for use with the RBF training algorithm that determines weights and bias values, and a Helpers class that contains utility display methods.

Class RadialNetwork isn't quite as complex as the program structure suggests because most of the class methods are helpers. Method Train performs the three-phase training process by calling helpers DoCentroids, DoWidths, and DoWeights. Private methods AvgAbsDist and DistinctIndices are helpers for DoCentroids. Method DoWeights uses private method Shuffle to process training data items in a different order each time through the iterative particle swarm optimization algorithm.

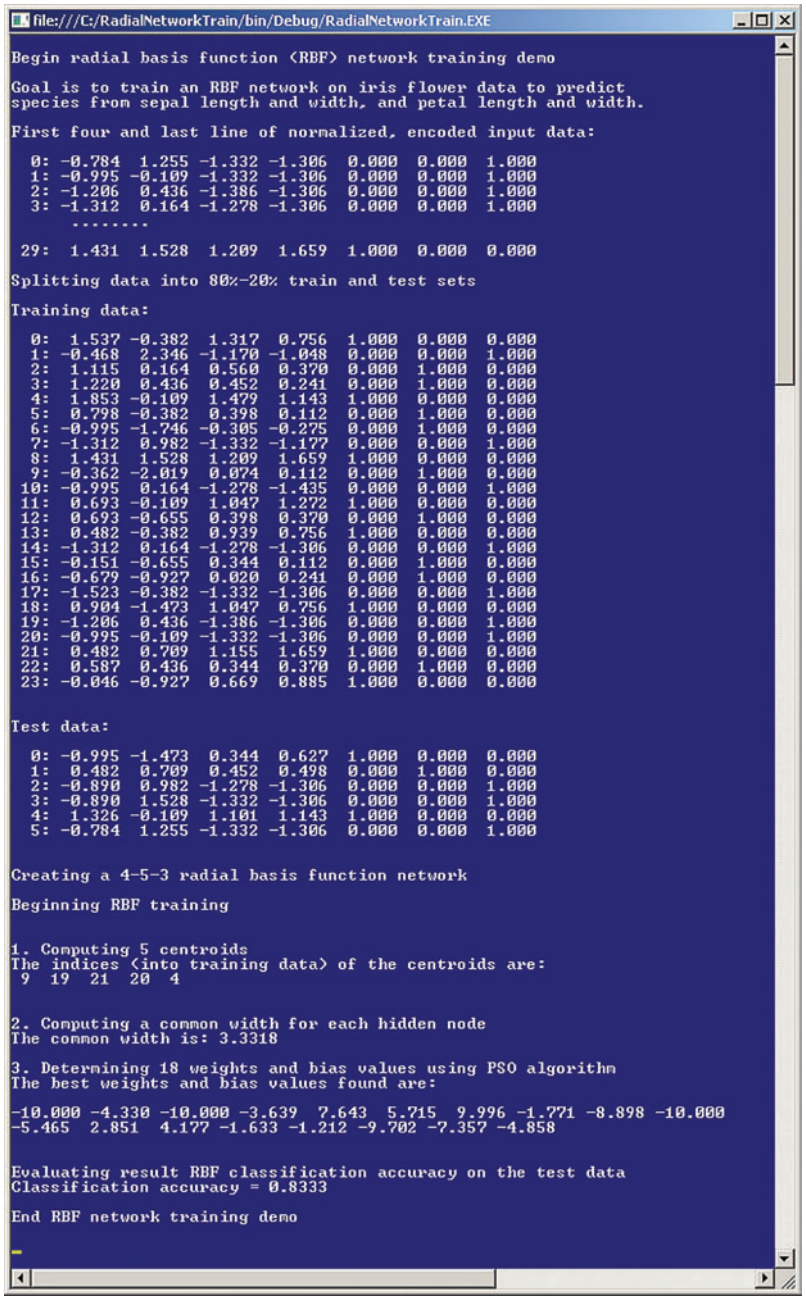


Figure 1 **A Radial Basis Function Network Demo Program**

The heart of the demo is fairly simple. First, the normalized and encoded data is set up:

```
double[][] allData = new double[30][];
allData[0] = new double[] { -0.784, 1.255, -1.332, -1.306, 0, 0, 1 };
allData[1] = new double[] { -0.995, -0.109, -1.332, -1.306, 0, 0, 1 };
// Etc.
allData[28] = new double[] { 0.904, -1.473, 1.047, 0.756, 1, 0, 0 };
allData[29] = new double[] { 1.431, 1.528, 1.209, 1.659, 1, 0, 0 };
```

Here the data is hardcoded for simplicity. In most scenarios your data will be stored in a text file or a SQL table. Next, the data is split into training and test subsets:

```
double[][] trainData = null;
double[][] testData = null;
int seed = 8;
GetTrainTest(allData, seed, out trainData, out testData);
```

The RBF network is instantiated:

```
int numInput = 4;
int numHidden = 5;
int numOutput = 3;
RadialNetwork rn = new RadialNetwork(numInput,
  numHidden, numOutput);
```

As mentioned in the previous section, the optimal number of hidden processing nodes must be determined essentially by trial and error. The network is trained:

```
int maxIterations = 100;
double[] bestWeights = rn.Train(trainData, maxIterations);
```

And, finally, the resulting model is evaluated:

```
rn.SetWeights(bestWeights);
double acc = rn.Accuracy(testData);
Console.WriteLine("Classification accuracy = " + acc.
ToString("F4"));
```

The bestWeights array holds the RBF weights and bias values as determined by the Train method. Method SetWeights loads these weights and bias values. You don't need to have the centroids and widths explicitly loaded because these values were set by method Train.

Radial Basis Function Network Input-Process-Output

To understand the RBF network training process, you need to understand the RBF network input-process-output mechanism. The diagram in **Figure 3** shows how the demo RBF network computes the outputs for test data item [1] = (0.482, 0.709, 0.452, 0.498) after the network has been trained. The input x-values are passed to each hidden node. Each hidden node computes its local output using its own centroid and width.

For example, the top-most hidden node's centroid is (-0.362, -2.019, 0.074, 0.112) and its width is 3.3318. The local outputs from each hidden node are then used to determine preliminary output values by computing a weighted sum of inputs, plus a bias value. For example, if hOutput[0] represents the local output of hidden node 0, then the preliminary output for the topmost output node is (hOutput[0] * w[0][0]) + (hOutput[1] * w[1][0]) + (hOutput[2] * w[2][0]) + (hOutput[3] * w[3][0]) + (hOutput[4] * w[4][0]) + bias[0] = -12.7999.

After the three preliminary output values have been computed, they're converted to final output values using a softmax function. The idea is to modify the preliminary output values so the final output values are all between 0.0 and 1.0, and sum to 1.0. This lets the output values be loosely interpreted as probabilities.

In **Figure 3**, the final outputs are (0.2897, 0.6865, 0.0237). Because the middle node has the highest value, it's interpreted as a 1, and the other two values are interpreted as 0, giving an inferred output of (0, 1, 0). Recall the test data is (0.482, 0.709, 0.452, 0.498, 0.000, 1.000, 0.000), where the first four values are inputs and the last three values are the target values, so the RBF network makes a correct prediction of the species (Iris versicolor in this case). The question now is: Where did the values for the RBF network's centroids, widths, weights and biases come from?

Determining RBF Network Centroids

The Train method of the RadialNetwork class is essentially a wrapper around three helper methods that do all the actual work:

```
public double[] Train(double[][] trainData, int maxIterations)
{
  DoCentroids(trainData);
  DoWidths(this.centroids);
  double[] bestWeights = DoWeights(trainData, maxIterations);
  return bestWeights;
}
```

Method DoCentroids determines representative input x-values. There are many possibilities here. One common approach is to use a k-means or k-medoids clustering algorithm that iteratively assigns and reassigns data items so similar data items are grouped together. When finished, each cluster will have a representative data member. You can use these as RBF centroids.

A different approach is to extract the x-values from randomly selected training data items. This is simple, but has the risk that bad centroids may be selected by chance.

The demo program uses what might be termed a lightweight clustering approach suggested by this pseudocode:

```
initialize maxDistance
intialize bestIndices
loop
  select numHidden random indices into train data
  compute an estimated distance between selected data items
  if estimated distance > maxDistance then
    set maxDistance = curr distance
    set bestIndices = curr indices
  end if
end loop
fetch the x-values in train data at bestIndices
store x-values into RBF centroids
```

The idea is best illustrated by example. Suppose the training data consists of the 24 items shown in **Figure 1**. Further suppose that the first time through the processing loop the four randomly selected indices are [0], [1], [2] and [3]. These correspond to:

```
0: ( 1.537, -0.382,  1.317,  0.756)
1: (-0.468,  2.346, -1.170, -1.048)
2: ( 1.115,  0.164,  0.560,  0.370)
3: ( 1.220,  0.436,  0.452,  0.241)
```

These are candidate centroids. The idea is to get representative x-values, which means you don't want values that are close together. So, you compute some measure of distance between these candidate centroids. Here, there are many possible approaches. The demo estimates an average distance between all possible pairs of candidate centroids by computing an average distance between adjacent pairs of candidates, instead of computing an average distance between all possible pairs. For this example, it computes the distances between candidates [0] and [1], between [1] and [2], and between [2] and [3].

A common approach to computing a distance is to use Euclidean distance, which is the square root of the sum of squared differences between values. (Note: The demo RBF network uses a Gaussian kernel, which uses Euclidean distance to compute hidden node local output values.) However, the demo program uses a variation of Manhattan distance, where distance is an average of the difference of absolute values. So, the distance between candidates [0] and [1] is:

```
d = abs(1.537 - (-0.468)) + . . . + abs(0.756 - (-1.048)) / 4
  = 2.256
```

The process of generating a set of candidate centroids and computing an estimated average distance for the set of candidates is repeated a specified number of times, and the set of candidates with the greatest estimated average distance is selected as the RBF centroid set.

Notice that determining RBF network centroids can be considered an unsupervised training technique because the target values (such as 0, 1, 0) in the training data aren't needed or used. This means centroids can be determined quickly. Also, RBF network widths, weights, and bias values can—in theory at least—be computed much more quickly than the roughly equivalent neural network weights and bias values. This gives RBF networks a potential advantage over neural networks (but, as you'll see, there's more to the story).

During the process of determining RBF network centroids, determining the candidate indices is an interesting subproblem. The demo program uses a clever algorithm called reservoir sampling. The idea is to pick the first possible n indices, then probabilistically replace the initial indices with the remaining possible indices:

```
private int[] DistinctIndices(int n, int range)
{
  // Reservoir sampling. assumes rnd exists
  int[] result = new int[n];
  for (int i = 0; i < n; ++i)
    result[i] = i;

  for (int t = n; t < range; ++t) {
    int m = rnd.Next(0, t + 1);
    if (m < n) result[m] = t;
  }
  return result;
}
```

Although the method is short, it's subtle. Alternatives include using a brute-force approach where random indices are generated and then checked to see if there are any duplicates.

Determining RBF Network Widths

The RBF network input-process-output mechanism requires a width value for each hidden node. There are many possibilities for determining the width values. The simplest approach, and the one used by the demo program, is to compute one common width, which all hidden processing nodes can use. Research in this area tends to be hazy and conclusions are sometimes contradictory. The demo program computes a common width as the average Euclidean distance between all possible pairs of centroids. In pseudocode:

```
sumOfDists = 0.0
for each pair of centroids
  accumulate Euclidean distance between curr pair
end loop
return accumulated sumOfDists / number of pairs
```

Figure 2 **Overall RBF Network Demo Program Structure**

```
using System;
namespace RadialNetworkTrain
{
  class RadialTrainProgram
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Begin radial basis function (RBF) network
        training demo");
      double[][] allData = new double[30][];
      allData[0] = new double[] { -0.784, 1.255, -1.332, -1.306, 0, 0, 1 };
      allData[1] = new double[] { -0.995, -0.109, -1.332, -1.306, 0, 0, 1 };
      // Etc.
      allData[28] = new double[] { 0.904, -1.473, 1.047, 0.756, 1, 0, 0 };
      allData[29] = new double[] { 1.431, 1.528, 1.209, 1.659, 1, 0, 0 };

      Console.WriteLine("First four and last line of normalized, encoded
        input data:");
      Helpers.ShowMatrix(allData, 4, 3, true, true);

      double[][] trainData = null;
      double[][] testData = null;
      int seed = 8; // Gives a good demo
      GetTrainTest(allData, seed, out trainData, out testData);
      Helpers.ShowMatrix(trainData, trainData.Length, 3, true, false);
      Helpers.ShowMatrix(testData, testData.Length, 3, true, false);

      int numInput = 4;
      int numHidden = 5;
      int numOutput = 3;
      RadialNetwork rn = new RadialNetwork(numInput, numHidden, numOutput);

      Console.WriteLine("Beginning RBF training");
      int maxIterations = 100; // Max for PSO
      double[] bestWeights = rn.Train(trainData, maxIterations);

      Console.WriteLine("Evaluating RBF accuracy on the test data");
      rn.SetWeights(bestWeights);

      double acc = rn.Accuracy(testData);
      Console.WriteLine("Classification accuracy = " + acc.ToString("F4"));

      Console.WriteLine("End RBF network training demo");
    }

    static void GetTrainTest(double[][] allData, int seed,
      out double[][] trainData, out double[][] testData) { . . }
  }

  public class RadialNetwork
  {
    private static Random rnd = null;
    private int numInput;
    private int numHidden;
    private int numOutput;
    private double[] inputs;
    private double[][] centroids;
    private double[] widths;
    private double[][] hoWeights;
    private double[] oBiases;
    private double[] outputs;

    public RadialNetwork(int numInput, int numHidden, int numOutput) { . . }
    private static double[][] MakeMatrix(int rows, int cols) { . . }

    public void SetWeights(double[] weights) { . . }
    public double[] GetWeights() { . . }

    private double MeanSquaredError(double[][] trainData,
      double[] weights) { . . }
    public double Accuracy(double[][] testData) { . . }
    private static int MaxIndex(double[] vector) { . . }

    public double[] ComputeOutputs(double[] xValues) { . . }
    private static double[] Softmax(double[] rawOutputs) { . . }

    public double[] Train(double[][] trainData, int maxIterations) { . . }
    private void DoCentroids(double[][] trainData) { . . }
    private static double AvgAbsDist(double[] v1, double[] v2,
      int numTerms) { . . }
    private int[] DistinctIndices(int n, int range) { . . }
    private void DoWidths(double[][] centroids) { . . }
    private double[] DoWeights(double[][] trainData, int maxIterations) { . . }
    private static double EuclideanDist(double[] v1, double[] v2,
      int numTerms) { . . }
    private static void Shuffle(int[] sequence) { . . }
  }

  public class Particle
  {
    // Implementation here
  }

  public class Helpers
  {
    // Implementation here
  }
}
```

Based on my experience, the effectiveness of RBF networks is extremely sensitive to the values used for hidden node widths. Research shows a width that's too small tends to over-fit the training data, leading to poor classification accuracy. A width that's too large tends to under-fit the data, which also leads to poor classification. If you experiment with the demo code by manually setting the values of the RBF network widths, you can see this effect in action.

Besides using the average distance between centroids, davg, for a common hidden nodes width value, research also suggests using (2 * davg), or (davg / sqrt(2 * numHidden)), and many other values. And instead of using a common width, there are many possibilities for computing different width values for each hidden node. In my opinion, the high sensitivity of RBF networks to width values, along with the related lack of convincing research results on how to best compute width values, are the major disadvantages of using RBF networks compared to alternatives such as neural networks and support vector machines.

Determining RBF Network Weights and Biases

After determining centroids and widths, the final step in training an RBF network is determining the values for the weights and biases. Theoretically, you can compute RBF network weights easily and quickly because, loosely speaking, there are n equations with n unknown values. So, standard numerical techniques can, in theory, be used to solve for the weight values.

Unfortunately, in practice, using standard techniques runs into many practical problems. For example, many standard techniques for solving systems of equations involve the use of matrix inversion. Matrix inversion can fail for many reasons.

Rather than use a deterministic but possibly brittle numerical technique to solve for RBF network weights exactly, the demo program uses particle swarm optimization to estimate the best values. PSO is a meta-heuristic based on coordinated group behavior, such as flocks of birds or schools of fish. In PSO, a particle has a position that represents a potential solution (the best set of weight values in this case). Each particle has a velocity that determines the particle's next position.

In PSO, a set of particles is created. In each simulated time tick, each particle moves to a new position based on the particle's current position and velocity, the best-known historical position of the particle, and the best-known historical position of any of the particles. Here's PSO in high-level pseudocode:

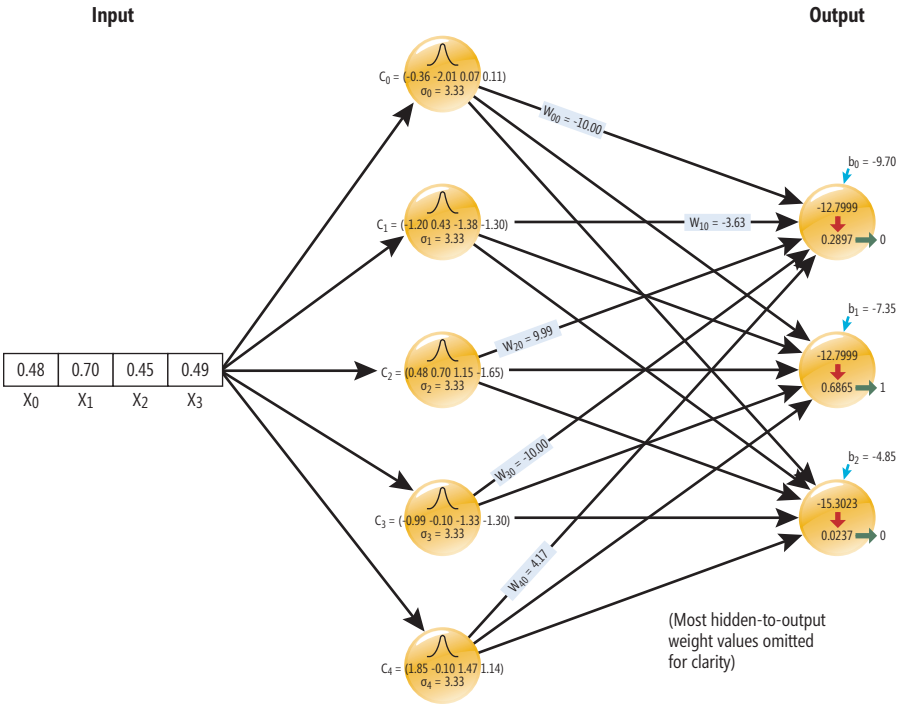


Figure 3 **Radial Basis Function Network Architecture**

```
set number of particles
set maxIterations
initialize all particles to random positions
loop maxIterations times
  for each particle
    update curr particle's velocity
    use new velocity to compute new position
    compute error for new position
    check if new particle best position
    check if new best position for all
particles
  end for
end loop
return best position found by any particle
```

PSO is a fascinating topic in its own right. You can learn more about it by reading my August 2011 article, "Particle Swarm Optimization" (msdn.microsoft.com/magazine/hh335067). PSO requires the specification of several free parameters, including weight constants that control the relative influence of a particle's current position, best historical position and best global historical position. PSO also requires specifying the number of particles, the maximum number of iterations and, optionally, an error threshold for early algorithm exit. You can experiment with these factors using the demo code.

In addition to PSO and traditional numerical techniques, there are many alternatives for finding RBF network weights, including simple gradient descent, and real-valued genetic algorithms. Although the theory of RBF networks has been studied fairly extensively, there are relatively few convincing research results on the comparative effectiveness of different training techniques.

Wrapping Up

The demo program code along with the explanation presented here should give you a solid foundation for investigating RBF networks. Although RBF networks are well-known in the research community, they don't seem to be used very often in the software developer community compared to alternatives such as neural network classifiers, naive Bayes classifiers, and logistic regression. One possible reason for this could be the scarcity of practical implementation examples. Another possible reason is the uncertainty surrounding fundamental RBF network factors, especially those related to the computation of RBF network widths. In my opinion, there's no solid research evidence to answer the question of whether RBF networks are more effective, less effective, or roughly equivalent to alternative machine-learning techniques. ■

Dr. James McCaffrey *works for Microsoft Research in Redmond, Wash. He has worked on several Microsoft products including Internet Explorer and Bing. He can be reached at jammc@microsoft.com.*

Thanks to the following technical expert for reviewing this article: *Kirk Olynyk (Microsoft Research)*

NEW HOSTING

> 300 Gbit/s network connection

Geo-redundancy

Web analytics

NetObjects Fusion® 2013

PHP 5.4

CDN

Free mode or safe mode

MySQL

SEO

Newsletter Tool

Over 140 apps

Drupal™, WordPress, Joomla!™, TYPO3, Magento® and many more...

Guaranteed Performance

Daily Backup

Mobile Website Builder

Adobe® Dreamweaver® CS5.5 included

COMPLETE PACKAGES FOR PROFESSIONALS STARTING AT

$ 1.99 /month*

MEMBER OF united internet

Call 1 (877) 461-2631
or buy online

1 TRIAL 30 DAY MONEY BACK GUARANTEE

1 MONTH FLEXIBLE PAYMENT OPTIONS

1 CALL SPEAK WITH AN EXPERT 24/7

1&1®

1and1.com

* 1&1 Web Hosting packages come with a 30 day money back guarantee, no minimum contract term, and no setup fee. Price reflects 36 month pre-payment option for 1&1 Basic package. Price goes to regular $5.99 per month price after 36 months. Some features listed only available with package upgrade or as add-on options. See website for full details.

Aspose.Total

Every Aspose component combined in one powerful suite.

Powerful File Format Components and Controls

Aspose.Words
DOC, DOCX, RTF, HTML, PDF, XPS & other document formats.

Aspose.Cells
XLS, XLSX, XLSM, XLTX, CSV, SpreadsheetML & image formats.

Aspose.BarCode
JPG, PNG, BMP, GIF, TIF, WMF, ICON & other image formats.

Aspose.Pdf
PDF, XML, XLS-FO, HTML, BMP, JPG, PNG & other image formats.

Aspose.Email
MSG, EML, PST, EMLX & other formats.

Aspose.Slides
PPT, PPTX, POT, POTX, XPS, HTML, PNG, PDF & other formats.

Aspose.Diagram
VSD, VSDX, VSS, VST, VSX & other formats.

... and many others!

Try Free Today!

- Aspose.Total for .NET
- Aspose.Total for Java
- Aspose.Total for SharePoint
- Aspose.Total for SSRS
- Aspose.Total for JasperReports
- Aspose.Total for Cloud

Entity Framework 6: The Ninja Edition

Julie Lerman

With the latest major release of Entity Framework, EF6, the Microsoft object-relational mapping (ORM) tool has reached new heights of "ninja-ness." It's no longer the country cousin to long-established .NET ORM tools. EF is all grown up, and it's winning over former die-hards.

Entity Framework has evolved through the choppy waters of its infancy, where it began as a tool focused on database developers—and inspired the wrath of agile developers within the .NET community. It learned how to get out of the way of application development and shifted to a Plain Old CLR Objects (POCOs) model, enabling testing and domain-focused software development without disenfranchising data-focused developers. It addressed performance issues and numerous concerns about the quality of generated code, and won over many database administrators (DBAs) along the way.

Beginning with EF 4.1, Microsoft recognized the complexity EF required and simplified access to its functionality by introducing the DbContext API. At the same time, because not everyone wants to use a designer or generated code, it provided the ability to build models with your own code. Along the way, there was another significant change that wasn't about features, syntax, code or performance. The EF team became more transparent and interactive with its community of users, and it began to provide feature releases more fluently rather than binding them to the Microsoft .NET Framework. This led to two advances after EF5 was released in 2012. First, all of the Entity Framework APIs were extracted from the .NET Framework and combined with the out-of-band feature APIs on which the team was also working. Second, the entire development effort moved to an open source model. EF6 has been developed publicly on entityframework.codeplex.com. Not only can you see what the team is doing via meeting notes, check-ins, and downloadable nightly builds, but you can also contribute source to EF6 (though with complete oversight by the EF team).

Keep in mind that EF6 is an evolution, not a revolution. Almost everything you already know about EF stays the same, such as how you build Entity Framework models and how you use EF in your applications. EF6 advances the ORM, but doesn't change how it fundamentally works. If you've invested in learning EF, that investment continues to pay off. EF6 doesn't come without some breaking changes—but these are limited to some namespace alterations that are easy enough to deal with if you're prepared. I'll point you to resources for guidance at the end of this article.

I think of EF6 features in a few categories:

1. Features that come for free: These are capabilities that are part of the core. You don't even have to know they're there to benefit from them, much less learn any new coding. This group includes features such as performance gains brought by a rewritten view-generation engine and query compilation modifications, stability granted by the ability of DbContext to use an already open connection, and a changed database setting for SQL Server databases created by Entity Framework.
2. Level-setting features: A major enhancement is that Code First now supports mapping to Stored Procedures, something that has been supported by models created in the designer. This feature has gotten a lot of coverage in Channel 9 videos (such as the one at bit.ly/16wL8fz) and in a detailed spec on the CodePlex site, so I won't repeat the information in this article.
3. Another change is more interesting. As I mentioned, with EF6, the EF APIs have been extracted from the .NET Framework; they're now completely encapsulated in the NuGet package. This means that certain features introduced with EF5—such as enum and spatial data support and improved performance—are no longer dependent

This article discusses:

- The move of Entity Framework away from .NET
- Performance improvements and stability
- New "ninja" features
- Contributions from the community

Technologies discussed:

Entity Framework 6, Microsoft .NET Framework

on .NET 4.5. So if you're using .NET 4with EF6, you can finally benefit from those features.

I'd also include the EF Designer in this category. It has been moved out of Visual Studio as of the 2013 edition, and instead provided as an extension to Visual Studio. For EF6, having the designer as an extension is a huge bonus. Going forward, the team will be able to add features directly to the designer, including those that are currently provided in the Entity Framework Power Tools. Separating the designer from Visual Studio allowed Microsoft to ship EF6 tooling for Visual Studio 2012 as well as Visual Studio 2013.

4. Ninja features: These are features you've craved ever since you got past the basic EF sample applications. There are many such features in EF6: support for asynchronous queries and saves, the return of custom Code First conventions, more extensibility using the new DbConfiguration type (which relies on the low-level EF6 IDbDependencyResolver), support for mocking in unit tests, configurable retries on spotty connections, and even more. You don't need to be a certified ninja to use these features—but you'll certainly feel like one when you do!

I also want to highlight a special category: EF6 contributions that came from community members. Unai Zorrilla added DbSet.AddRange and RemoveRange, the ability to customize pluralization and the handy DbChangeTracker.HasChanges method. He's also working on other cool features for a future iteration of EF. Erik Jensen, a SQL Server Compact (SQLCE) MVP, contributed SQLCeFunctions, which are similar to the SqlFunctions for using SQL Server functions in LINQ to Entities queries. The greatly improved speed of EF view generation—most dramatic for large, complex models—was driven by Alireza Haghshenas and a CodePlex member named VSavenkov. It's also possible now to define custom migration operations, thanks to Iñaki Elcoro, aka iceclow on CodePlex. (Rowan Miller of the EF team wrote some blog posts about this feature; the first is at bit.ly/ZBUOw1.) A full list of contributors can be found in the team blog post, "EF6 RTM Available," at bit.ly/1gmDE6D.

In this article, I'll drill into some of the less-publicized topics and point you to existing resources to learn more about the others.

A Version History page on the MSDN Data Developer Center (bit.ly/1gCTOnz) lists all of the features, each with a sentence or two of detail and some with links to more information.

It Just Works: Performance Improvements and Stability

Performance is the bane of many a software project and there has been plenty of criticism of the performance of Entity Framework since its inception. However, each iteration of EF has brought vast improvements in this area.

One of the biggest drags on performance is the startup time involved with the first use of a context in an application process. You can do a lot to improve that startup time, though. Hopefully you've already learned these tricks from my own writing or other resources, such as the MSDN doc on performance considerations at bit.ly/3D6AiC.

A startup step that often hampers performance is the view generation of mapping views, where EF creates the relevant SQL to query against each of the entity sets in the model. These views get leveraged as your app runs so that for certain queries, EF doesn't have to work out the SQL on the fly. View generation happens whether you created your model with the EF Designer or with Code First. You can pre-generate these views and compile them into the application to save time.

For large, complex models, view generation was especially time-consuming. This process has been revamped for EF6, improving the speed dramatically, whether you pre-generate the views or let this happen at run time. Note that there was a bug in the EF 6.0.0 release that hindered this feature, but it was corrected in EF 6.0.1, which was released on the same day and is (at the time of writing) the default package that you'll get via NuGet. Additionally, the way EF uses those generated views at run time has been enhanced, improving query execution time. View generation on small or simple models was never an issue. But plenty of organizations have models with hundreds of entities that also include inheritance, relationships and other complications. Those organizations will benefit greatly from this change.

On another performance note, see the guidance about using Ngen against the Entity Framework assembly in the announcement blog post for the release of EF6 at bit.ly/1gmDE6D.

Faster LINQ Contains Compilation The EF team continues to tweak how queries are created, and one change the team has highlighted is how queries using LINQ Contains are compiled. To be clear, it's the performance of the compilation process that has improved. The generated SQL hasn't changed, so the execution of the query in the database isn't affected.

SQL Server Database Creation One of the stability improvements in EF6 is related to database creation. Both the Model First and Code First workflows can create a database for you. If that database is SQL Server, EF is now aligned with a "best practice" for SQL Server databases, which is to configure the database's READ_COMMITTED_SNAPSHOT setting to ON. This means that, by default, the database will create a snapshot of itself every time a change is made. Queries will be performed on the snapshot while updates are performed on the actual database. I wrote about this feature in a recent blog post, "What's that Read_Committed_Snapshot Transaction Support for EF6 About Anyway?" at bit.ly/14FDpZI.

Reuse Open Connections Finally, a frustrating limitation has been removed: EF6 lets you execute context calls on an open DbConnection. In the past, if you explicitly opened a connection before executing the EF command that used that connection, or you attempted to reuse a connection that had already been opened by another context call, an exception would be thrown with the message "Entity Connection can only be constructed with a closed DbConnection." Now, EF6 is more than happy to let you reuse an already open connection.

Ninja Enhancements

Async Support I explored a handful of new features—Async querying, SaveChanges and custom conventions—in "Playing with the EF6 Alpha," in my March 2013 Data Points column (msdn.microsoft.com/magazine/jj991973).

Async support brings the .NET 4.5 Await and Async pattern to the LINQ query execution methods for EF, giving you FirstAsync,

FirstOrDefaultAsync, SingleAsync, SingleOrDefaultAsync, ToListAsync, ForEachAsync and more. To see the full list, check System.Data.Entity.QueryableExtensions. DbSet gained FindAsync and DbContext gained SaveChangesAsync. Since that article, not much has changed, so you can take a look at it to get more details. In addition, Microsoft created some walk-throughs and an interesting detailed specification, which you can get to from the version history page I mentioned earlier.

Custom Code Conventions I also wrote about custom Code First conventions in that article—another ninja feature, for sure. The EF team worked on this for the initial release of Code First, but it was holding up the release and the team was forced to set it aside—to the disappointment of many developers.

Suppose you have a common mapping you want to apply as a general rule to your entities or properties. Now you can define it as a convention rather than having to specify the mapping individually for each entity or property in your model, and it will be applied across the board. For example, if you want each string property to be represented in your database as 50 characters, instead of whatever default your database provider uses, you can specify this rule as a convention. The conventions leverage the Code First Fluent API, so building conventions should feel familiar if you've configured mappings this way:

```
modelBuilder.Properties<String>().Configure(p => p.HasMaxLength(50))
```

Now, every string in this model will be mapped to a database column of 50 characters. As with the Fluent or annotations configurations, you can specify conventions for properties or entities, and control inheritance mappings. Affecting relationships via convention is a less common and more complex task handled by model-based conventions. For more information, see bit.ly/1gAqcMq. You can also use a convention as a data annotation. There's a hierarchy for executing conventions when Code First is building its model. By default, built-in conventions run first and custom conventions run afterward. But you can force a custom convention to precede a built-in convention. Check out "Custom Code First Conventions" at bit.ly/14dgOCP for examples.

Connection Resiliency If a connection is dropped while EF is attempting to execute a query or save changes, you now have the ability to tell EF to retry. Though dropped connections can be a problem on corporate intranets, connection resiliency has proven to be quite useful in helping apps that connect to the cloud. The retries can be configured using IDbConnectionStrategy. The SQL Server provider included with EF specifies a default:SqlServerExecutionStrategy, which has an error message suggesting that you tune the strategy for exceptions thrown by transient connections.

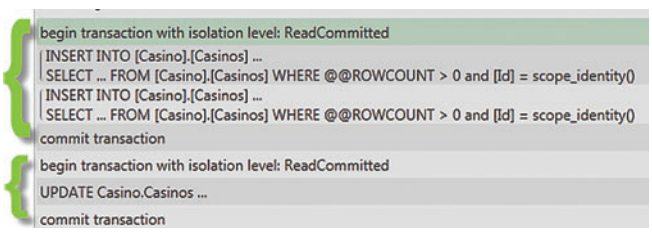


Figure 1 **Commands from Separate Context Calls Wrapped in Their Own Transactions**

Another, SqlAzureExecutionStrategy, is tuned for connections to Windows Azure SQL Database.

The simplest way to specify a strategy is with the new DbConfiguration class, which makes it easy to configure how a particular database provider should behave. The following tells EF to use SQLAzureExecutionStrategy for SqlClient:

```
SetExecutionStrategy (SqlProviderServices.ProviderInvariantName,
  () => new SqlAzureExecutionStrategy());
```

Not only are the connection strategies configurable, but you can also create your own as well as suspend them programmatically as needed. EF team member Miller shows you how to suspend in his blog post at bit.ly/14gPM1y.

I tested SqlAzureExecutionStrategy using a trick suggested by another EF team member, Glenn Condron. To trigger the particular transient connection fault error codes that this strategy looks for, I used the new EF6 command interception feature to throw a transient connection error. Then I ran a test whose output showed that when I set the execution strategy, the query was retried five times after the initial failure. There's a great comment on my blog post about this feature from a developer who says his company is already witnessing the benefits of this feature (bit.ly/HaqMA0).

There's also an interesting algorithm that ensures retries on different threads don't all execute at the same time.

Share DbTransactions and DbConnections I hope you're aware by now that EF always uses a DbTransaction by default for calls made to the database. For example, when calling SaveChanges, a DbTransaction is created before the first command is sent to the database. EF then sends all necessary insert, update and delete commands to the database, and finally commits the transaction. If one command fails, all of the previously executed commands are rolled back.

You've always had the ability to override that default behavior by spinning up a TransactionScope to wrap the EF call and any other calls (not necessarily database- or EF-related) that need to be in the same transaction. An addition to EF6 now lets a single DbTransaction be responsible for multiple database calls. Note that you'll still need to use a TransactionScope if you want to include non-database logic within the transaction or distributed transactions for calls to different databases.

The key to sharing DbTransactions with EF6 is a new BeginTransaction method that returns a reference to the current DbTransaction and a UseTransaction method.

This code demonstrates the default behavior:

```
//code to create two new casinos, "casino1" & "casino2"
var context = new CasinoSlotsModel();
context.Casinos.AddRange(new[] { casino1, casino2 });
context.SaveChanges();
context.Database.ExecuteSqlCommand
  ("Update Casino.Casinos set rating= " + (int) casino.Rating)
```

My profiler shows a transaction being used around each context call—one to SaveChanges, which triggered two inserts, and one to ExecuteSqlCommand, which triggered the update—as shown in **Figure 1**.

Now I'll modify the code to share a transaction, wrapping the SaveChanges and ExecuteSqlCommand calls. I'll use the new DbContext.Database.BeginTransaction to explicitly instantiate a System.Data.Entity.DbContextTransaction, and open a connection if necessary (keep in mind that there's a similar command with

DbContext.Database.Connection, but that returns a System.Data.Common.DbTransaction, which can't be shared by the EF commands):

```
using (var tx = context.Database.BeginTransaction()) {
  try  {
    context.SaveChanges();
    context.Database.ExecuteSqlCommand
      ("Update Casino.Casinos set rating= " + (int) casino.Rating);
    tx.Commit();
  }
  catch (Exception)  {
    tx.Rollback();
  }
}
```

You can see in **Figure 2** that all of the commands are wrapped in the same transaction.

There's another new feature called UseTransaction. You can spin up a DbTransaction and then use it for ADO.NET calls and, with DbContext.Database.UseTransaction, execute EF calls even from separate context instances within the same transaction. You can see an example of this at bit.ly/1aEMIuX.

It's also possible to explicitly reuse an open connection, because EF can now create an EntityConnection (something the ObjectContext does in the background) with an already opened connection. Also, a context won't close a connection that it didn't open itself. I wrote a simple test in which I open a context's connection before executing a call from the context:

```
[TestMethod]
public void ContextCanCreateEntityConnectionWithOpenConnection()
{
  using (var context = new CasinoSlotsModel())
  {
    context.Database.Connection.Open();
    Assert.IsNotNull(context.Casinos.ToList());
  }
}
```

When executing the ToList call, the DbContext will create an ObjectContext instance that in turn will create an EntityConnection. It then uses the EntityConnection to open the DbConnection and closes the DbConnection when the call is complete with all results returned. Running this in EF5 causes an exception ("EntityConnection can only be constructed with a closed DbConnection"), but it succeeds in EF6 because of the change in behavior. This change will allow you to reuse connections in scenarios where you want to have more control over the state of connection. The specs suggest scenarios "such as sharing a connection between components where you cannot guarantee the state of the connection."

AddRange and RemoveRange As mentioned earlier, AddRange and RemoveRange are contributions from community member Zorrilla. Each method takes as its parameter an enumerable of a single entity type. In the first code sample in the sharing DbTransactions section, I used AddRange when I passed in an array of Casino instances:

```
context.Casinos.AddRange(new[] { casino1, casino2 });
```

These methods execute much faster than adding or removing a single object at a time because, by default, Entity Framework calls DetectChanges in each Add and Remove method. With the Range methods, you can handle multiple objects while DetectChanges is called only once, improving performance dramatically. I've tested this using five, 50, 500, 5,000 and even 50,000 objects and, at least in my scenario, there's no limit to the size of the array—and it's impressively fast! Keep in mind that this improvement is only relevant in getting the context to act on the objects, and has no bearing on SaveChanges. Calling SaveChanges still executes just one database command at a time. So while you can quickly add 50,000 objects into a context, you'll still get 50,000 insert commands executed individually when you call SaveChanges—probably not something you want to do in a real system.

On the flip side of this, there were long discussions about implementing support for bulk operations without requiring objects to be tracked by EF (bit.ly/16tMHw4), and for batch operations to enable sending multiple commands together in a single call to the database (bit.ly/PegT17). Neither feature made it into the initial EF6 release, but both are important and slated for a future release.

Less Interference with Your Coding Style In .NET, it's possible to override the System.Object.Equals method to define your system's rules for equality. Entity Framework, however, has its own way of determining the equality of tracked entities, and this relies on identity. If you've overwritten Equals (and the GetHashCode method that Equals is dependent upon), you can trip up the change-tracking behavior of Entity Framework. Petar Paar demonstrates this problem very clearly in his blog post at bit.ly/GJcohQ. To correct this problem, EF6 now uses its own Equals and GetHashCode logic to perform change-tracking tasks, ignoring any custom Equals and GetHashCode logic you may have written. However, you can still make explicit calls to your own custom methods in your domain logic. This way, the two approaches can live in harmony.

If you're focused on graphs and aggregates, you may want to nest types within other types. But the Code First model builder wasn't able to discover nested types to create entities or complex types in a model. **Figure 3** shows an example of nested type: Address. I'll use Address only in the Casino type, so I've nested it in the Casino class. I've also created Address as a Domain-Driven Design (DDD) value object because it doesn't need its own identity. (See my October 2013 Data Points column, "Coding for Domain-Driven Design: Tips for Data-Focused Devs, Part 3," at msdn.microsoft.com/magazine/dn451438 for more about DDD value objects.)

I used the EF Power Tools to get a visual representation of the model, and in **Figure 4**, I show the resulting Casino entity rendered when using EF5 and EF6. EF5 didn't recognize the nested type and didn't include Address or the dependent properties (PhysicalAddress and MailingAddress) in the model. But EF6 was able to detect the nested type, and you can see the Address fields were represented in the model.

The same under-the-covers changes that enable the nested types also solve another problem. This is the problem raised by having multiple types with the same names under different namespaces in the same project. Previously, when EF read the metadata from the EDMX and looked for the matching type in the assembly, it didn't pay attention to namespaces.

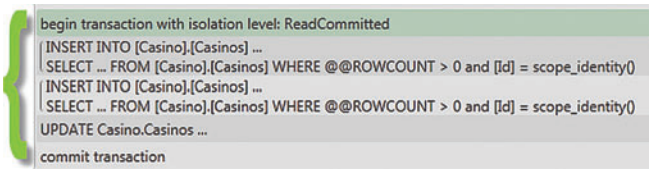


Figure 2 **Commands from All Context Calls in a Single Transaction**

Thank You!

We want to extend our deepest thanks and appreciation to all our loyal customers who took the time to vote for their favorite DevExpress products in this year's Visual Studio Magazine Reader's Choice Awards.

Learn more and download your free trial

devexpress.com/try

DevExpress™

A common scenario where this caused issues was including non-EF representations of entities in the same project as the model that contained the entities. So I might have this code-generated PokerTable class from my model:

```
namespace CasinoEntities
{
  public partial class PokerTable
  {
    public int Id { get; set; }
    public string Description { get; set; }
    public string SerialNo { get; set; }
  }
}
```

I might also have this DTO class that's not part of my model but is in the same project:

```
namespace Casino.DataTransferObjects
{
  public class PokerTable
  {
    public int Id { get; set; }
    public string Description { get; set; }
    public string SerialNo { get; set; }
  }
}
```

When the project targets EF5, you'll see the following error when building the project:

> The mapping of CLR type to EDM type is ambiguous because multiple CLR types match the EDM type 'PokerTable.' Previously found CLR type 'MyDtos.PokerTable,' newly found CLR type 'EDMXModel.DTOs.PokerTable.'

You'll see this error at design time with an EDMX. With Code First, if you had the same scenario—two matching classes with the same name but different namespaces, and one of those classes in the model—you'd see the problem at run time when the model builder begins interpreting the Code First model.

This problem has long been a source of frustration. I've read that message and asked my computer: "Do you not see the different namespaces? Hello?" I also have a collection of e-mails from friends, clients and other developers who have experienced this problem.

EF6 will now recognize the namespaces and allow this scenario. You can read more about the internals that enabled these two changes in the blog post by Arthur Vickers at bit.ly/Wi1rZA.

Figure 3 **My Casino Class with a Nested Address Type**

```
public class Casino()
{
  //...other Casino properties & logic

  public Address PhysicalAddress { get; set; }
  public Address MailingAddress { get; set; }

  public class Address:ValueObject<Address>
  {
    protected Address(){    }
    public Address(string streetOrPoBox, string city,
                   string state,string postalCode)
    { City = city;
      State = state;
      PostalCode = postalCode;
      StreetOrPoBox = streetOrPoBox; }

    public string StreetOrPoBox { get; private set; }
    public string City { get; private set; }
    public string State { get; private set; }
    public string PostalCode { get; private set; }
  }
}
```

Move Context Configuration into Code There are a number of settings you can apply in your app.config or web.config files to define how your context should work. For example, you can specify a database initialization or migration or a default database provider. My method of adding these has often been copying and pasting because the settings are too hard to remember and involve a lot of strings. Now, with EF6, you can declare many context configurations in code using the DbConfiguration class. I played with this in the March Data Points column, but encountered a bug caused by a race condition, which I reported and has since been fixed. So I'll revisit DbConfigurations (also referred to as code-based configuration) now. Note that there's plenty of room for confusion when talking about configuration mappings for Code First versus DbConfiguration settings versus DbMigrationConfigurations for defining how database migration will work when you change your model. DbConfiguration is targeted at DbContext settings.

DbConfiguration depends on another super-ninja, low-level feature of EF6: support for dependency resolution, similar to the IDependencyResolver used in ASP.NET MVC and Web API. Dependency resolution allows you to use the Service Locator pattern with the Inversion of Control (IoC) pattern in your code, allowing EF6 to choose from a hierarchy of available objects that implement a common interface. In this case the root interface is IDbDependencyResolver. While EF6 includes a number of DbConfigurations that allow it to discover and place precedence on the context settings, it's possible to leverage dependency resolution to add new features to EF as well. I'll stick to showing some of the configurations and point you to the feature spec at bit.ly/QKtvCr for more details about the IDbDependencyResolver.

You can use DbConfiguration to specify familiar context rules as well as settings that are new to EF6. **Figure 5** shows a sample configuration class that includes a host of different settings EF will use in place of its default behaviors. Notice the settings are placed in the class constructor.

SetDefaultConnectionFactory supplants the DefaultConnectionFactory tag you may already be using in your config file in the entityframework section. SetDatabaseInitializer replaces specifying an initializer or migration configuration in your config file or at application startup. I show two examples, though one is commented out. SetExecutionStrategy lets you specify what to do if your connection drops in the middle of EF queries or other execution commands. SetPluralizationService exposes another new feature of EF6: the ability to create custom pluralizations. I'll explain more in a bit.

There are a slew of other ways to affect the context with these built-in dependency resolvers. The MSDN document, "IDbDependencyResolver Services" (bit.ly/13Aojso), lists all of the resolvers that are available with DbConfiguration. Dependency resolution is also used to help provider writers solve certain issues when they need to inject rules and logic into how the context interacts with the provider.

Query and Command Interception I failed to mention CustomDbConfiguration's use of AddInterceptor. DbConfiguration lets you do more than shove in IDbDependencyResolvers. Another new feature of EF6 is the ability to intercept queries and commands. When intercepting queries and commands, you now have access to the generated SQL that's about to be sent to the database and

Windows. Web. Mobile.
Your next great app starts here.

DevExpress .NET controls, frameworks and libraries were built with you in mind. Built for those who demand the highest quality and expect the best performance... for those who require reliable tools engineered to meet today's needs and address tomorrow's requirements.

Experience the DevExpress Difference today and download your free 30-day trial and let's build great apps, together.

WinForms · ASP.NET · WPF · Silverlight · Windows 8 XAML · HTML JS · Reporting · DevExtreme Mobile

Learn more and download your free trial

devexpress.com/try

DevExpress™

results that are coming back from those commands. You can use this information to log the SQL commands or even modify them and tell EF to use the updated commands. Although I enjoyed playing with this feature, I'll save some space by pointing you to Part 3 of Arthur Vickers' three-part blog series on it: "EF6 SQL Logging – Part 3: Interception building blocks" (bit.ly/19om5du), which has links back to Part 1 ("Simple Logging") and Part 2 ("Changing the content/formatting").

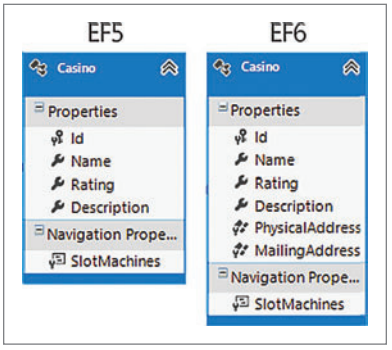


Figure 4 **Unlike EF5, EF6 Sees the Nested Address Type and Includes the Dependent Properties**

Customize EF Pluralization Before talking about the ability to customize EF pluralization, I want to be sure you understand its default behavior. EF uses its internal pluralization service for three tasks:

1. In Database First, it ensures that entities have a singularized name. So if your database table is named People, it will become Person in the model.
2. In the EF Designer for Database or Model First, it creates pluralized EntitySet names (the basis of DbSets) based on the Entity name. For example, the service will be sure that the EntitySet name for the Person entity is People. It doesn't just use the table name if you created the model using Database First.
3. In Code First, where you explicitly name the DbSets, EF uses the service to infer table names. If you're starting with a class named Person, convention will assume that your database table is named People.

The service doesn't work like spelling dictionaries, where you can provide a textual list of custom spellings. Instead it uses internal rules. Aside from the occasional anomaly (I had fun with the entity name Rhinoceros early on), the biggest problem with the service is that the rules are based on English.

For EF6, Zorrilla created the IPluralizationService interface so you can add your own logic. Once you've created a custom service, you can plug it in with DbConfiguration as shown earlier.

Currently this customization will only work with the third case in the previous list: when Code First is inferring the table names. As of the initial EF6 release, the customization can't be applied to the designer.

There are two ways to use the service. You can start with a base service—either the EnglishPluralizationService in EF or one that someone else already built—and then override the Singularize or Pluralize methods to add your own rules. Additionally you can specify a pair of words in a CustomPluralizationEntry class and attach the class to an existing service. Zorrilla demonstrates the CustomPluralizationEntry in his blog post (bit.ly/161JrD6).

Look for a future Data Points column in which I'll show an example of adding rules (not just word pairs) for pluralization, with a demonstration of the effect on Code First database mappings.

Figure 5 **Sample Configuration Class**

```
public class CustomDbConfiguration : DbConfiguration
{
  public CustomDbConfiguration()
  {
    SetDefaultConnectionFactory(new LocalDbConnectionFactory("v11.0"));
    SetDatabaseInitializer
      (new MigrateDatabaseToLatestVersion<CasinoSlotsModel, Configuration>());
    //SetDatabaseInitializer(new MyInitializer());
    SetExecutionStrategy("System.Data.SqlClient",
      () => new SqlAzureExecutionStrategy());
    AddInterceptor(new NLogEfCommandInterceptor());
    SetPluralizationService(new CustomPluralizationService());
  }
}
```

More Code First Goodies

There are a handful of new features targeting Code First that I haven't covered yet—specifically, Code First migrations. For reasons of space, I'll highlight them here and follow up with a more in-depth look in the January 2014 Data Points column:

- Ability to create migration scripts that can check to see which have been run already so you can fix up a database from any migration point.
- More control over the Migrations_History table to account for different database providers.
- Ability to specify default schema for database mapping instead of always defaulting to dbo.
- Ability of migrations to handle different DbContexts that target the same database.
- Ability of ModelBuilder to add multiple EntityTypeConfigurations at once rather than using one line of code for each.

Go Get Ninja!

In my opinion, the most important thing to remember about EF6 is that it adds great features to what already exists in EF. If you're moving projects from EF5 to EF6, you should be aware of some namespace changes. The team has detailed guidance for making this move at bit.ly/17eCB4U. Other than that, you should feel confident about using EF6, which is now the current stable version of Entity Framework that's distributed through NuGet. Even if you don't intend to use any of the ninja features right away, remember that you'll still benefit from increased performance as described earlier in this article. I'm most excited about the ninja features, though, and grateful to the developers from the community who've added to EF6 as well.

EF will continue to evolve. While the first release of EF6 was timed to coincide with the release of Visual Studio 2013, EF 6.1 and versions beyond that are already in the works. You can follow their progress on the CodePlex site. ■

Julie Lerman *is a Microsoft MVP, .NET mentor and consultant who lives in the hills of Vermont. You can find her presenting on data access and other Microsoft .NET topics at user groups and conferences around the world. She blogs at thedatafarm.com/blog and is the author of "Programming Entity Framework" (2010) as well as a Code First edition (2011) and a DbContext edition (2012), all from O'Reilly Media. Follow her on Twitter at twitter.com/julielerman and see her Pluralsight courses at juliel.me/PS-Videos.*

Thanks to the following technical expert for reviewing this article: *Rowan Miller (Microsoft)*

CORS Support in ASP.NET Web API 2

Brock Allen

Cross-origin resource sharing (CORS) is a World Wide Web Consortium (W3C) specification (commonly considered part of HTML5) that lets JavaScript overcome the same-origin policy security restriction imposed by browsers. The same-origin policy means that your JavaScript can only make AJAX calls back to the same origin of the containing Web page (where "origin" is defined as the combination of host name, protocol and port number). For example, JavaScript on a Web page from http://foo.com can't make AJAX calls to http://bar.com (or to http://www.foo.com, https://foo.com or http://foo.com:999, for that matter).

CORS relaxes this restriction by letting servers indicate which origins are allowed to call them. CORS is enforced by browsers but must be implemented on the server, and the most recent release of ASP.NET Web API 2 has full CORS support. With Web API 2, you can configure policy to allow JavaScript clients from a different origin to access your APIs.

CORS Basics

To use the new CORS features in Web API, it's helpful to understand the details of CORS itself, because the Web API implementation is true to the specification. These details might seem pedantic now, but they'll be useful later to understand the available settings in Web API—and when you're debugging CORS they'll help you fix problems faster.

The general mechanics of CORS are such that when JavaScript is attempting to make a cross-origin AJAX call the browser will "ask" the server if this is allowed by sending headers in the HTTP request (for example, Origin). The server indicates what's allowed by returning HTTP headers in the response (for example, Access-Control-Allow-Origin). This permission check is done for each distinct URL the client invokes, which means different URLs can have different permissions.

In addition to the origin, CORS lets a server indicate which HTTP methods are allowed, which HTTP request headers a client can send, which HTTP response headers a client can read, and if the browser is allowed to automatically send or receive credentials (cookies or authorization headers). Additional request and response headers indicate which of these features are allowed. These headers are summarized in **Figure 1** (note that some of the features have no header sent in the request—only the response).

Browsers can ask the server for these permissions in two different ways: simple CORS requests and preflight CORS requests.

Simple CORS Requests Here's an example of a simple CORS request:

```
POST http://localhost/WebApiCorsServer/Resources/ HTTP/1.1
Host: localhost
Accept: */*
Origin: http://localhost:55912
Content-Type: application/x-www-form-urlencoded; charset=UTF-8

value1=foo&value2=5
```

And the response:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Access-Control-Allow-Origin: http://localhost:55912
Content-Length: 27

{"Value1":"foo","Value2":5}
```

This article discusses:

- Understanding the details of cross-origin resource sharing (CORS)
- Using the basic CORS framework
- Different approaches to developing a dynamic CORS policy
- Techniques to debug failed cross-origin AJAX calls

Technologies discussed:

ASP.NET Web API 2, JavaScript

The request is a cross-origin request from http://localhost:55912 to http://localhost, and the browser adds an Origin HTTP header in the request to indicate the calling origin to the server. The server responds with an Access-Control-Allow-Origin response header indicating that this origin is allowed. The browser enforces the server's policy, and the JavaScript will receive its normal success callback.

Figure 1 **CORS HTTP Headers**

Permission/Feature	Request Header	Response Header
Origin	Origin	Access-Control-Allow-Origin
HTTP method	Access-Control-Request-Method	Access-Control-Allow-Method
Request headers	Access-Control-Request-Headers	Access-Control-Allow-Headers
Response headers		Access-Control-Expose-Headers
Credentials		Access-Control-Allow-Credentials
Cache preflight response		Access-Control-Max-Age

The server can either respond with the exact origin value from the request or a value of "*" indicating any origin is allowed. If the server hadn't allowed the calling origin, then the Access-Control-Allow-Origin header would simply be absent and the calling JavaScript's error callback would be invoked.

Note that with a simple CORS request the call on the server is still invoked. This can be surprising if you're still learning about CORS, but this behavior is no different from a scenario where the browser had constructed a <form> element and made a normal POST request. CORS doesn't prevent the call from being invoked on the server; rather, it prevents the calling JavaScript from receiving the results. If you want to prevent the caller from invoking the server, then you'd implement some sort of authorization in your server code (possibly with the [Authorize] authorization filter attribute).

The preceding example is known as a simple CORS request because the type of AJAX call from the client was either a GET or a POST; the Content-Type was one of application/x-www-form-urlencoded, multipart/form-data, or text/plain; and there were no additional request headers sent. If the AJAX call was another HTTP method, the Content-Type was some other value or the client wanted to send additional request headers, then the request would be considered a preflight request. The mechanics of preflight requests are slightly different.

> With Web API 2, you can configure policy to allow JavaScript clients from a different origin to access your APIs.

Preflight CORS Requests If an AJAX call isn't a simple request, then it requires a preflight CORS request, which is simply an additional HTTP request to the server to obtain permission. This preflight request is made automatically by the browser and uses the OPTIONS HTTP method. If the server responds successfully to the preflight request and grants permission, then the browser will perform the actual AJAX call the JavaScript is attempting to make.

If performance is a concern (and when isn't it?), then the outcome of this preflight request can be cached by the browser by including the Access-Control-Max-Age header in the preflight response. The value contains the number of seconds for which the permissions can be cached.

Here's an example of a preflight CORS request:

```
OPTIONS http://localhost/WebApiCorsServer/Resources/1 HTTP/1.1
Host: localhost
Access-Control-Request-Method: PUT
Origin: http://localhost:55912
Access-Control-Request-Headers: content-type
Accept: */*
```

And the preflight response:

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: http://localhost:55912
Access-Control-Allow-Methods: PUT
Access-Control-Allow-Headers: content-type
Access-Control-Max-Age: 600
```

Here's the actual AJAX request:

```
PUT http://localhost/WebApiCorsServer/Resources/1 HTTP/1.1
Host: localhost
Content-Length: 27
Accept: application/json, text/javascript, */*; q=0.01
Origin: http://localhost:55912
Content-Type: application/json

{"value1":"foo","value2":5}
```

And the AJAX response:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Access-Control-Allow-Origin: http://localhost:55912
Content-Length: 27

{"Value1":"foo","Value2":5}
```

Notice in this example a preflight CORS request is triggered because the HTTP method is PUT and the client needs to send the Content-Type header to indicate that the request contains application/json. In the preflight request (in addition to Origin) the Access-Control-Request-Method and Access-Control-Request-Headers request headers are used to ask for permission for the type of HTTP method and the additional header the client wishes to send.

The server granted permission (and set a preflight cache duration) and then the browser allowed the actual AJAX call. If the server didn't grant permission to any of the requested features, then the corresponding response header would've been absent, the AJAX call wouldn't have been made and the JavaScript error callback would've been invoked instead.

The preceding HTTP requests and responses were made using Firefox. If you were to use Internet Explorer, then you'd notice an additional Accept header being requested. If you were to use Chrome, you'd see both Accept and Origin additionally requested. Interestingly, you won't see Accept or Origin in the Access-Control-Allow-Headers, as the specification says they're implied and can be omitted (which Web API does). It's a point of debate if Origin and Accept actually need to be requested, but given how these browsers work today, your Web API CORS policy will most likely need to include them. It's unfortunate that browser vendors don't seem to be consistent in their reading of the specification.

Response Headers It's easy to give a client permission to access response headers using the Access-Control-Expose-Headers response header. Here's an example of an HTTP response that allows the calling JavaScript to access the custom response header "bar":

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Access-Control-Allow-Origin: http://localhost:55912
Access-Control-Expose-Headers: bar
bar: a bar value
Content-Length: 27

{"Value1":"foo","Value2":5}
```

> Possibly the most confusing aspect of CORS has to do with credentials and authentication.

The JavaScript client can simply use the XMLHttpRequest getResponseHeader function to read the value. Here's an example using jQuery:

```
$.ajax({
  url: "http://localhost/WebApiCorsServer/Resources/1",
  // other settings omitted
}).done(function (data, status, xhr) {
  var bar = xhr.getResponseHeader("bar");
  alert(bar);
});
```

Credentials and Authentication Possibly the most confusing aspect of CORS has to do with credentials and authentication. Generally, authentication with Web APIs can be done either with a cookie or with an Authorization header (there are other ways, but these two are the most common). In normal browser activity, if one of these has been previously established, then the browser will implicitly pass these values to the server on subsequent requests. With cross-origin AJAX, though, this implicit passing of the values must be explicitly requested in JavaScript (via the withCredentials flag on the XMLHttpRequest) and must be explicitly allowed in the server's CORS policy (via the Access-Control-Allow-Credentials response header).

Here's an example of a JavaScript client setting the withCredentials flag with jQuery:

```
$.ajax({
  url: "http://localhost/WebApiCorsServer/Resources/1",
  xhrFields: {
    withCredentials: true
  }
  // Other settings omitted
});
```

The withCredentials flag does two things: If the server issues a cookie, the browser can accept it; if the browser has a cookie, it can send it to the server.

Here's an example of the HTTP response allowing credentials:

```
HTTP/1.1 200 OK
Set-Cookie: foo=1379020091825
Access-Control-Allow-Origin: http://localhost:55912
Access-Control-Allow-Credentials: true
```

The Access-Control-Allow-Credentials response header does two things: If the response has a cookie, the browser can accept it; and if the browser sent a cookie on the request, the JavaScript client can receive the results of the call. In other words, if the client sets withCredentials, then the client will only see a success callback in the JavaScript if the server (in the response) allows credentials. If withCredentials was set and the server doesn't allow credentials, the client won't get access to the results and the client error callback will be invoked.

The same set of rules and behaviors apply if the Authorization header is used instead of cookies (for example, when using Basic or Integrated Windows authentication). An interesting note about using credentials and the Authorization header: The server doesn't have to explicitly grant the Authorization header in the Access-Control-Allow-Headers CORS response header.

Note that with the Access-Control-Allow-Credentials CORS response header, if the server issues this header, then the wildcard value of "*" can't be used for Access-Control-Allow-Origin. Instead the CORS specification requires the explicit origin to be used. The Web API framework handles all of this for you, but I mention it here because you might notice this behavior while debugging.

There's an interesting twist to this discussion of credentials and authentication. The description up to this point has been for the scenario where the browser is *implicitly* sending credentials. It's possible for a JavaScript client to *explicitly* send credentials (again, typically via the Authorization header). If this is the case, then none of the aforementioned rules or behaviors related to credentials applies.

For this scenario, the client would explicitly set the Authorization header on the request and wouldn't need to set withCredentials on the XMLHttpRequest. This header would trigger a preflight request and the server would need to allow the Authorization header with the Access-Control-Allow-Headers CORS response header. Also, the server wouldn't need to issue the Access-Control-Allow-Credentials CORS response header.

Figure 2 **Applying the EnableCors Attribute to Action Methods**

```
public class ResourcesController : ApiController
{
  [EnableCors("http://localhost:55912", // Origin
              null,                     // Request headers
              "GET",                    // HTTP methods
              "bar",                    // Response headers
              SupportsCredentials=true  // Allow credentials
  )]
  public HttpResponseMessage Get(int id)
  {
    var resp = Request.CreateResponse(HttpStatusCode.NoContent);
    resp.Headers.Add("bar", "a bar value");
    return resp;
  }

  [EnableCors("http://localhost:55912",       // Origin
              "Accept, Origin, Content-Type", // Request headers
              "PUT",                          // HTTP methods
              PreflightMaxAge=600             // Preflight cache duration
  )]
  public HttpResponseMessage Put(Resource data)
  {
    return Request.CreateResponse(HttpStatusCode.OK, data);
  }

  [EnableCors("http://localhost:55912",       // Origin
              "Accept, Origin, Content-Type", // Request headers
              "POST",                         // HTTP methods
              PreflightMaxAge=600             // Preflight cache duration
  )]
  public HttpResponseMessage Post(Resource data)
  {
    return Request.CreateResponse(HttpStatusCode.OK, data);
  }
}
```

GdPicture.NET 10

- Document viewing, processing, printing and scanning (TWAIN & WIA).
- Reading, writing and converting vector and raster images in more than 90 formats, PDF included.
- OMR, OCR, barcode reading and writing (linear & 2D).
- Annotations for image and PDF within Windows and Web applications.
- Color detection engine for image and PDF compression.

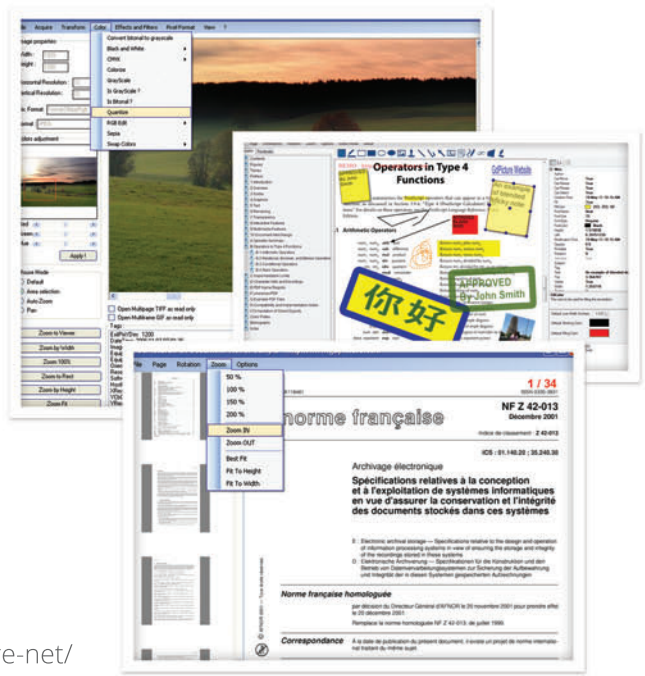
And much more ...

All-In-One Document Imaging SDK

Royalty-Free Document Imaging Toolkits for .NET and COM/ActiveX

GdPicture.NET 10 Plugins

- NEW Color detection
- Full managed PDF support
- Full annotations support for PDF and images
- OCR
- Forms processing
- JBIG2 encoding
- 1D and 2D barcode reading and writing



Try GdPicture.NET 10 FREE for 30 days

www.componentsource.com/products/gdpicture-net/

Here's what that client code would look like to explicitly set the Authorization header:

```
$.ajax({
  url: "http://localhost/WebApiCorsServer/Resources/1",
  headers: {
    "Authorization": "Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3Mi..."
  }
  // Other settings omitted
});
```

Here's the preflight request:

```
OPTIONS http://localhost/WebApiCorsServer/Resources/1 HTTP/1.1
Host: localhost
Access-Control-Request-Method: GET
Origin: http://localhost:55912
Access-Control-Request-Headers: authorization
Accept: */*
```

Here's the preflight response:

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Access-Control-Allow-Headers: authorization
```

Explicitly setting a token value in the Authorization header is a safer approach to authentication because you avoid the possibility of cross-site request forgery (CSRF) attacks. You can see this approach in the new Single-Page Application (SPA) templates in Visual Studio 2013.

Now that you've seen the basics of CORS at the HTTP level, I'll show you how to use the new CORS framework to emit these headers from Web API.

CORS Support in Web API 2

The CORS support in Web API is a full framework for allowing an application to define the permissions for CORS requests. The framework revolves around the concept of a policy that lets you specify the CORS features to be allowed for any given request into the application.

First, in order to get the CORS framework, you must reference the CORS libraries from your Web API application (they're not referenced by default from any of the Web API templates in Visual Studio 2013). The Web API CORS framework is available via NuGet as the Microsoft.AspNet.WebApi.Cors package. If you're not using NuGet, it's also available as part of Visual Studio 2013, and you'll need to reference two assemblies: System.Web.Http.Cors.dll and System.Web.Cors.dll (on my machine these are located in C:\Program Files (x86)\Microsoft ASP.NET\ASP.NET Web Stack 5\Packages).

Next, to express the policy, Web API provides a custom attribute class called EnableCorsAttribute. This class contains properties for the allowed origins, HTTP methods, request headers, response headers and whether credentials are allowed (which model all of the details of the CORS specification discussed earlier).

Figure 3 **Using Explicit Values for HTTP Methods**

```
[EnableCors("*", "*", "PUT, POST")]
public class ResourcesController : ApiController
{
  public HttpResponseMessage Put(Resource data)
  {
    return Request.CreateResponse(HttpStatusCode.OK, data);
  }

  public HttpResponseMessage Post(Resource data)
  {
    return Request.CreateResponse(HttpStatusCode.OK, data);
  }

  // CORS not allowed because DELETE is not in the method list above
  public HttpResponseMessage Delete(int id)
  {
    return Request.CreateResponse(HttpStatusCode.NoContent);
  }
}
```

Finally, in order for the Web API CORS framework to process CORS requests and emit the appropriate CORS response headers, it must look at every request into the application. Web API has an extensibility point for such interception via message handlers. Appropriately, the Web API CORS framework implements a message handler called CorsMessageHandler. For CORS requests, it will consult the policy expressed in the attribute for the method being invoked and emit the appropriate CORS response headers.

EnableCorsAttribute The EnableCorsAttribute class is how an application can express its CORS policy. The EnableCorsAttribute class has an overloaded constructor that can accept either three or four parameters. The parameters (in order) are:

1. List of origins allowed
2. List of request headers allowed
3. List of HTTP methods allowed
4. List of response headers allowed (optional)

There's also a property for allowing credentials (SupportsCredentials) and another for specifying the preflight cache duration value (PreflightMaxAge).

Figure 2 shows an example of applying the EnableCors attribute to individual methods on a controller. The values being used for the various CORS policy settings should match the CORS requests and responses that were shown in the prior examples.

Notice each of the constructor parameters is a string. Multiple values are indicated by specifying a comma-separated list (as is specified for the allowed request headers in **Figure 2**). If you wish to allow all origins, request headers or HTTP methods, you can use a "*" as the value (you must still be explicit for response headers).

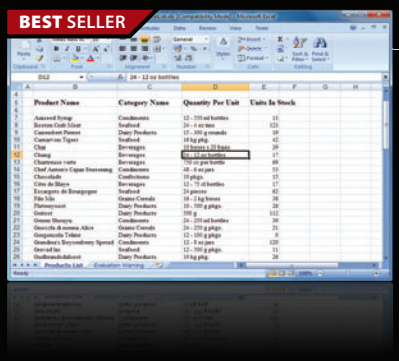
In addition to applying the EnableCors attribute at the method level, you can also apply it at the class level or globally to the application. The level at which the attribute is applied configures CORS for all requests at that level and below in your Web API code. So, for example, if applied at the method level, the policy will only apply to requests for that action, whereas if applied at the class level, the policy will be for all requests to that controller. Finally, if applied globally, the policy will be for all requests.

Following is another example of applying the attribute at the class level. The settings used in this example are quite permissive because the wildcard is used for the allowed origins, request headers and HTTP methods:

```
[EnableCors("*", "*", "*")]
public class ResourcesController : ApiController
{
  public HttpResponseMessage Put(Resource data)
  {
    return Request.CreateResponse(HttpStatusCode.OK, data);
  }

  public HttpResponseMessage Post(Resource data)
  {
    return Request.CreateResponse(HttpStatusCode.OK, data);
  }
}
```

If there's a policy at multiple locations, the "closest" attribute is used and the others are ignored (so the precedence is method, then class, then global). If you've applied the policy at a higher level

BEST SELLER



Aspose.Total for .NET | from $2,449.02

ASPOSE Your File Format Experts

Every Aspose .NET component in one package.

- Programmatically manage popular file formats including Word, Excel, PowerPoint and PDF
- Work with charts, diagrams, images, Project plans, emails, barcodes, OCR, and document management in .NET applications
- Common uses also include mail merging, adding barcodes to documents, building dynamic reports on the fly and extracting text from PDF files

NEW VERSION

GdPicture.NET | from $4,600.56

GdPicture imaging technologies

All-in-one AnyCPU document-imaging and PDF toolkit for .NET and ActiveX.

- Document viewing, processing, printing, scanning, OMR, OCR, Barcode Recognition
- Annotate image and PDF within your Windows & Web applications
- Read, write and convert vector & raster images in more than 90 formats, including PDF
- Color detection engine for image and PDF compression
- 100% royalty-free and world leading Imaging SDK

BEST SELLER

ComponentOne Studio Enterprise 2013 v2 | from $1,315.60

ComponentOne a division of GrapeCity

.NET Tools for the Professional Developer: Windows, HTML5/Web, and XAML.

- Hundreds of UI controls for all .NET platforms including grids, charts, reports and schedulers
- Visual Studio 2013 and Windows 8.1 support
- 40+ UI widgets built with HTML5, jQuery, CSS3, and SVG
- New TouchToolkit for touch enabling WinForms apps
- Royalty-free deployment and distribution

BEST SELLER

Help & Manual Professional | from $583.10

ec software

Easily create documentation for Windows, the Web and iPad.

- Powerful features in an easy accessible and intuitive user interface
- As easy to use as a word processor, but with all the power of a true WYSIWYG XML editor
- Single source, multi-channel publishing with conditional and customized output features
- Output to HTML, WebHelp, CHM, PDF, ePUB, RTF, e-book or print
- Styles and Templates give you full design control

We accept purchase orders.
Contact us to apply for a credit account.

US Headquarters
ComponentSource
650 Claremore Prof Way
Suite 100
Woodstock
GA 30188-5188
USA

European Headquarters
ComponentSource
30 Greyfriars Road
Reading
Berkshire
RG1 1PE
United Kingdom

Asia / Pacific Headquarters
ComponentSource
3F Kojimachi Square Bldg
3-3 Kojimachi Chiyoda-ku
Tokyo
Japan
102-0083

Sales Hotline - US & Canada:
(888) 850-9911
www.componentsource.com

GSA Schedule Contract GS-35F-0188R

but then wish to exclude a request at a lower level, you can use another attribute class called DisableCorsAttribute. This attribute, in essence, is a policy with no permissions allowed.

If you have other methods on the controller where you don't want to allow CORS, you can use one of two options. First, you can be explicit in the HTTP method list, as shown in **Figure 3**. Or you can leave the wildcard, but exclude the Delete method with the DisableCors attribute, as shown in **Figure 4**.

CorsMessageHandler The CorsMessageHandler must be enabled for the CORS framework to perform its job of intercepting requests to evaluate the CORS policy and emit the CORS response headers. Enabling the message handler is typically done in the application's Web API configuration class by invoking the EnableCors extension method:

```
public static class WebApiConfig
{
  public static void Register(HttpConfiguration config)
  {
    // Other configuration omitted

    config.EnableCors();
  }
}
```

If you wish to provide a global CORS policy, you can pass an instance of the EnableCorsAttribute class as a parameter to the EnableCors method. For example, the following code would configure a permissive CORS policy globally within the application:

```
public static class WebApiConfig
{
  public static void Register(HttpConfiguration config)
  {
    // Other configuration omitted

    config.EnableCors(new EnableCorsAttribute("*", "*", "*"));
  }
}
```

As with any message handler, the CorsMessageHandler can alternatively be registered per-route rather than globally.

So that's it for the basic, "out of the box" CORS framework in ASP.NET Web API 2. One nice thing about the framework is that it's extensible for more dynamic scenarios, which I'll look at next.

Customizing Policy

It should be obvious from the earlier examples that the list of origins (if the wildcard isn't being used) is a static list compiled into the Web API code. While this might work during development or for specific scenarios, it isn't sufficient if the list of origins (or other permissions) needs to be determined dynamically (say, from a database).

Fortunately, the CORS framework in Web API is extensible such that supporting a dynamic list of origins is easy. In fact, the framework is so flexible that there are two general approaches for customizing the generation of policy.

Custom CORS Policy Attribute One approach to enable a dynamic CORS policy is to develop a custom attribute class that can generate the policy from some data source. This custom attribute class can be used instead of the EnableCorsAttribute class provided by Web API. This approach is simple and retains the fine-grained feel of being able to apply an attribute on specific classes and methods (and not apply it on others), as needed.

To implement this approach, you simply build a custom attribute similar to the existing EnableCorsAttribute class. The main focus is the ICorsPolicyProvider interface, which is responsible for creating an instance of a CorsPolicy for any given request. **Figure 5** contains an example.

Figure 4 **Using the DisableCors Attribute**

```
[EnableCors("*", "*", "*")]
public class ResourcesController : ApiController
{
  public HttpResponseMessage Put(Resource data)
  {
    return Request.CreateResponse(HttpStatusCode.OK, data);
  }

  public HttpResponseMessage Post(Resource data)
  {
    return Request.CreateResponse(HttpStatusCode.OK, data);
  }

  // CORS not allowed because of the [DisableCors] attribute
  [DisableCors]
  public HttpResponseMessage Delete(int id)
  {
    return Request.CreateResponse(HttpStatusCode.NoContent);
  }
}
```

The CorsPolicy class has all the properties to express the CORS permissions to grant. The values used here are just an example, but presumably they could be populated dynamically from a database query (or from any other source).

Custom Policy Provider Factory The second general approach to building a dynamic CORS policy is to create a custom policy provider factory. This is the piece of the CORS framework that obtains the policy provider for the current request. The default implementation from Web API uses the custom attributes to discover the policy provider (as you saw earlier, the attribute class

Figure 5 **A Custom CORS Policy Attribute**

```
[AttributeUsage(AttributeTargets.Class | AttributeTargets.Method,
                AllowMultiple = false)]
public class EnableCorsForPaidCustomersAttribute :
  Attribute, ICorsPolicyProvider
{
  public async Task<CorsPolicy> GetCorsPolicyAsync(
    HttpRequestMessage request, CancellationToken cancellationToken)
  {
    var corsRequestContext = request.GetCorsRequestContext();
    var originRequested = corsRequestContext.Origin;
    if (await IsOriginFromAPaidCustomer(originRequested))
    {
      // Grant CORS request
      var policy = new CorsPolicy
      {
        AllowAnyHeader = true,
        AllowAnyMethod = true,
      };
      policy.Origins.Add(originRequested);
      return policy;
    }
    else
    {
      // Reject CORS request
      return null;
    }
  }

  private async Task<bool> IsOriginFromAPaidCustomer(
    string originRequested)
  {
    // Do database look up here to determine if origin should be allowed
    return true;
  }
}
```


REPORTING
ActiveReports 7
With royalty-free licensing, create:
Form-based reports, such as invoices & insurance documents
Transaction reports, such as sales & accounting
Analytical reports, such as sales & budget analysis
& portfolio analysis
Download your free trial @
www.componentone.com
ComponentOne
a division of GrapeCity
© 2013 GrapeCity, inc. All rights reserved. All other product and brand names are trademarks and/or registered trademarks of their respective holders.

Figure 6 **A Custom Policy Provider Factory**

```
public class DynamicPolicyProviderFactory : ICorsPolicyProviderFactory
{
  public ICorsPolicyProvider GetCorsPolicyProvider(
    HttpRequestMessage request)
  {
    var route = request.GetRouteData();
    var controller = (string)route.Values["controller"];
    var corsRequestContext = request.GetCorsRequestContext();
    var originRequested = corsRequestContext.Origin;

    var policy = GetPolicyForControllerAndOrigin(
      controller, originRequested);
    return new CustomPolicyProvider(policy);
  }

  private CorsPolicy GetPolicyForControllerAndOrigin(
   string controller, string originRequested)
  {
    // Do database lookup to determine if the controller is allowed for
    // the origin and create CorsPolicy if it is (otherwise return null)
    var policy = new CorsPolicy();
    policy.Origins.Add(originRequested);
    policy.Methods.Add("GET");
    return policy;
  }
}

public class CustomPolicyProvider : ICorsPolicyProvider
{
  CorsPolicy policy;
  public CustomPolicyProvider(CorsPolicy policy)
  {
    this.policy = policy;
  }

  public Task<CorsPolicy> GetCorsPolicyAsync(
    HttpRequestMessage request, CancellationToken cancellationToken)
  {
    return Task.FromResult(this.policy);
  }
}
```

itself was the policy provider). This is another pluggable piece of the CORS framework, and you'd implement your own policy provider factory if you wanted to use an approach for policy other than custom attributes.

The attribute-based approach described earlier provides an implicit association from a request to a policy. A custom policy provider factory approach is different from the attribute approach because it requires your implementation to provide the logic to match the incoming request to a policy. This approach is more coarse-grained, as it's essentially a centralized approach for obtaining a CORS policy.

Figure 6 shows an example of what a custom policy provider factory might look like. The main focus in this example is the implementation of the ICorsPolicyProviderFactory interface and its GetCorsPolicyProvider method.

The main difference in this approach is that it's entirely up to the implementation to determine the policy from the incoming request. In **Figure 6**, the controller and origin could be used to query a database for the policy values. Again, this approach is the most flexible, but it potentially requires more work to determine the policy from the request.

To use the custom policy provider factory, you must register it with Web API via the SetCorsPolicyProviderFactory extension method in the Web API configuration:

```
public static class WebApiConfig
{
  public static void Register(HttpConfiguration config)
  {
    // Other configuration omitted

    config.EnableCors();
    config.SetCorsPolicyProviderFactory(
      new DynamicPolicyProviderFactory());
  }
}
```

Debugging CORS

A few techniques come to mind to debug CORS if (and when) your cross-origin AJAX calls aren't working.

Client Side One approach to debugging is to simply use your HTTP debugger of choice (for example, Fiddler) and inspect all HTTP requests. Armed with the knowledge gleaned earlier about the details of the CORS specification, you can usually sort out why a particular AJAX request isn't being granted permission by inspecting the CORS HTTP headers (or lack thereof).

Another approach is to use your browser's F12 developer tools. The console window in modern browsers provides a useful error message when an AJAX calls fails due to CORS.

Server Side The CORS framework itself provides detailed trace messages using the tracing facilities of Web API. As long as an ITraceWriter is registered with Web API, the CORS framework will emit messages with information about the policy provider selected, the policy used, and the CORS HTTP headers emitted. For more information on Web API tracing, consult the Web API documentation on MSDN.

A Highly Requested Feature

CORS has been a highly requested feature for some time now, and finally it's built in to Web API. This article focuses heavily on the details of CORS itself, but that knowledge is crucial in implementing and debugging CORS. Armed with this knowledge, you should be able to easily utilize the CORS support in Web API to allow cross-origin calls in your applications. ■

Community Contributions in Action

ASP.NET Web API is an open source framework and is part of a larger set of open source frameworks collectively called the ASP.NET Web Stack, which also includes MVC, Web Pages and others.

These frameworks are used to build the ASP.NET platform and are curated by the ASP.NET team at Microsoft. As curator of an open source platform, the ASP.NET team welcomes community contributions, and the cross-origin resource sharing (CORS) implementation in Web API is one such contribution.

It was originally developed by Brock Allen as part of the thinktecture IdentityModel security library (**thinktecture.github.io**).

Brock Allen *is a consultant specializing in the Microsoft .NET Framework, Web development and Web-based security. He's also an instructor for the training company DevelopMentor, associate consultant for thinktecture GmbH & Co. KG, a contributor to thinktecture open source projects and a contributor to the ASP.NET platform. You can reach him at his Web site, brockallen.com, or e-mail him at brockallen@gmail.com.*

Thanks to the following technical expert for reviewing this article:
Yao Huan Lin (Microsoft)

Spread Controls for

Windows Forms & ASP.NET

WPF & Silverlight

WinRT

Spread Studio for .NET contains our new cross-platform spreadsheet controls for Windows Forms, ASP.NET, WPF, WinRT, and Silverlight in one amazing package.

Experience for yourself:

Excel®-like grid. Native Microsoft® Excel® Compatibility - same look & feel for your users

Flexible & familiar spreadsheet architecture, advanced charting & powerful formula library

Create financial modeling & risk analysis, budgeting, insurance, scientific applications & more

NEW

Spread Studio for .NET

ComponentOne®
a division of GrapeCity®

Download your free trial @
www.componentone.com

Cross-Browser, Coded UI Testing with Visual Studio 2013

Damian Zapart

Web-based solutions have become popular in the past few years because they provide easy access to users all around the world. Users also like them because of their convenience. Users don't need to install a separate application; with browsers alone they can connect to their accounts from any device connected to the Internet. However, for both software developers and testers, the fact that a user can choose any Web browser presents a problem—you must test a solution against multiple browsers. In this article, I'll demonstrate how this problem might be resolved in a simple way by creating coded UI test cases that will execute against any modern browser, using only C#.

This article uses Visual Studio 2013 Release Candidate. Information is subject to change.

This article discusses:

- New testing features in Visual Studio 2013
- Creating a new Coded UI Project
- Finding and extracting HTML controls from a Web page
- Working with multiple browsers
- Testing a user scenario

Technologies discussed:

Visual Studio 2013, C#

Code download available at:

archive.msdn.microsoft.com/mag201312Testing

The New Visual Studio

A few years ago, when Visual Studio 2010 was released, one of its most interesting features was the ability to test the UI of Web-based solutions. However, at the time, there were some limitations of using this technology; for example, the only Web browser supported was Internet Explorer. Moreover, testing the UI relied on recording user actions on the Web site and then replaying them to simulate real user actions, which many developers found unacceptable.

The new Visual Studio 2013, available as a release candidate (RC), brings a host of improvements in many different areas, ranging from new IDE features to an extended test framework (a long list of changes in the RC version is available at bit.ly/1bBryTZ). From my perspective, two new features are particularly interesting. First, you can now test the UI of not only Internet Explorer (including Internet Explorer 11) but also all other modern browsers, such as Google Chrome and Mozilla Firefox. Second, and even more crucial from a test development point of view, is what Microsoft calls "configurable properties for coded UI tests on the browser." Basically, this new functionality defines a set of search criteria for UI elements. I'll describe these features in more detail later in this article.

System Under Test

Those two new features are what I'll use to create cross-browser, fully coded UI tests. For my system under test (SUT), I want a public, well-known, Web-based application, so I chose Facebook. I want to cover two basic user scenarios. The first scenario is a positive test case that will display a profile page after successful login. The second is a

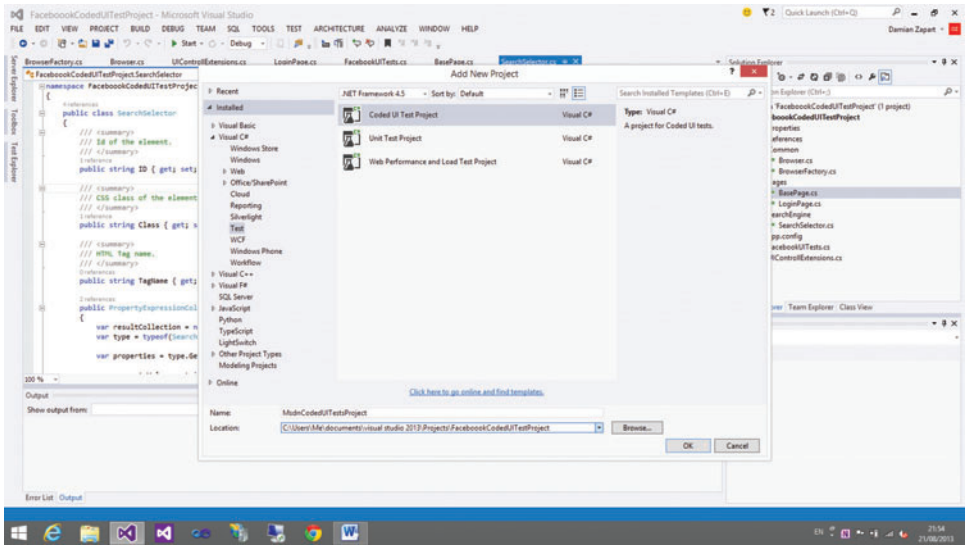


Figure 1 **Creating a New Coded UI Test Project**

negative test case in which I enter invalid user credentials and try to log in. In this case, I expect some error message in the user response.

There are a few challenges I'll need to resolve. First, the correct browser needs to be launched (based on the test configuration), and it must be able to provide access to a specific URL. Second, during run time, a specific control element must be extracted from the HTML document to provide input for the simulated user. Wherever needed, values must be entered for the control and the correct buttons clicked to submit an HTML form to the server. The code should also be able to handle a response from the server, validate it and, finally, close the browser when the test case's cleanup method).

Before Coding

Before I start coding I need to prepare my environment, which is pretty straightforward. First I need to download Visual Studio 2013 RC from bit.ly/137Sg3U. By default, Visual Studio 2013 RC lets you create coded UI tests just for Internet Explorer, but that's not what I'm interested in; I want to create tests for all modern browsers. Of course, there will be no compilation errors as long as I indicate in my code that the tests should run against browsers other than Internet Explorer, but an unhandled exception will be thrown during run time. Later on I'll show how to change the browser as well. To avoid problems during coding, I need to download and install a Visual Studio extension called "Selenium components for Coded UI Cross-Browser Testing" (bit.ly/Sd7Pgw), which will let me execute tests on any browser installed on my machine.

Jump into the Code

With everything in place I can now demonstrate how to create a new Coded UI Project. Open Visual Studio 2013 and click on File | New Project | Templates | Visual C# | Tests | Coded UI Test Project. Enter the project name and press OK to see the new solution, as shown in **Figure 1**.

The solution is essentially divided into three strongly connected groups, as shown in **Figure 2**. The first group contains a Pages namespace, which includes the BasePage class, from which ProfilePage and LoginPage derive. These classes expose properties and logic for operating on the page currently displayed in the browser. Such an approach helps split the test case implementations from browser-specific operations such as searching for a control by Id. The test cases operate directly on the properties and functions exposed by the page classes.

In the second group I put all extensions (UIControlExtensions), selectors (SearchSelector) and browser-related common classes (BrowserFactory, Browser). This subset of objects stores the implementation logic of the HTML elements search engine (which I'll describe shortly). I also added my own browser-related objects, which help in running test cases against the correct Web browser. The last group contains the test file (FacebookUITests) with the implementations of the test cases. Those test cases never operate on the browser directly; instead, they use the panel classes.

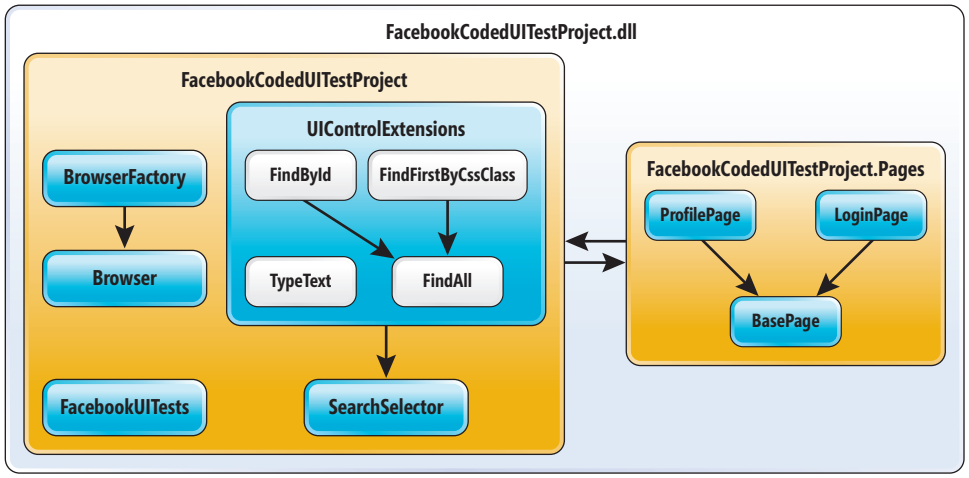


Figure 2 **Coded UI Test Solution Diagram**

An important part of my project is the HTML controls search engine, so my first step is to create a static class called UIControlExtensions, which contains the implementation logic for finding and extracting specific controls from the currently opened Web browser page. To make the coding easier—and to be able to reuse it later in the project—I don't want to have to initialize instances of this class every time I need to use it, so

Figure 3 **Searching for HTML Controls**

```
private static ReadOnlyCollection<T> FindAll<T>(this UITestControl
control, SearchSelector selectorDefinition) where T : HtmlControl, new()
{
  var result = new List<T>();
    T selectorElement = new T { Container = control };
      selectorElement.SearchProperties.AddRange(
        selectorDefinition.ToPropertyCollection());

    if (!selectorElement.Exists)
      {
        Trace.WriteLine(string.Format(
          "Html {0} element doesn't exist for given selector {1}.",
            typeof(T).Name, selectorDefinition),"UI CodedTest");
        return result.AsReadOnly();
      }

    return selectorElement
        .FindMatchingControls()
        .Select(c => (T)c).ToList().AsReadOnly();
      }
}
```

I'm going to implement it as an extension method that will extend the built-in UITestControl type. Additionally, any extension functions I implement will be generic. They must derive from HtmlControl (the base class for all UI controls in the Coded UI Test Framework) and must contain a default parameter-less constructor. I want this function to be generic because I intend to search only for specific control types (see the list of available HTML types at bit.ly/1aiB5eW).

I'll pass search criteria to my function via the selectorDefinition parameter, of SearchSelector type. The SearchSelector class is a simple type, but it's still very useful. It exposes a number of properties, such as ID or Class, which can be set from another function and later be transformed, using reflection, to the PropertyExpressionCollection class (bit.ly/18lvmnd). Next, that property collection can be used as a filter to extract just the small subset of HTML controls that match the given criteria. Later, the generated property collection is assigned to the SearchProperties property (bit.ly/16C20iS) of the generic object, which lets it call the Exists property and FindMatchingControls function. Keep in mind that Coded UI Test Framework algorithms won't search for specific controls on the whole page by default, and will process all children and subchildren only on the extended UITestControl. This helps improve the performance of the test cases because search criteria can be applied to just a small subset of the HTML document; for example, to all children of some DIV container, as shown in **Figure 3**.

I've implemented the core of the search control engine, but the function FindAll<T> requires a lot of code and knowledge to get it working properly—you have to specify the search parameters, check whether an item exists, and so on. That's why I decided to make it private and expose two other functions instead:

```
public static T FindById<T>(this UITestControl control, string
controlId) where T : HtmlControl, new()
public static T FindFirstByCssClass<T>(this UITestControl control,
string className, bool contains = true) where T : HtmlControl, new()
```

Figure 4 **The BasePage Class**

```
public abstract class BasePage : UITestControl
{
  protected const string BaseURL = "https://www.facebook.com/";

  /// <summary>
  /// Gets URL address of the current page.
  /// </summary>
  public Uri PageUrl{get; protected set;}

  /// <summary>
  /// Store the root control for the page.
  /// </summary>
  protected UITestControl Body;

  /// <summary>
  /// Gets current browser window.
  /// </summary>
  protected BrowserWindow BrowserWindow { get; set; }

  /// <summary>
  /// Default constructor.
  /// </summary>
  public BasePage()
  {
    this.ConstructUrl();
  }
  /// <summary>
  /// Builds derived page URL based on the BaseURL and specific page URL.
  /// </summary>
  /// <returns>A specific URL for the page.</returns>
  protected abstract Uri ConstructUrl();

  /// <summary>
  /// Verifies derived page is displayed correctly.
  /// </summary>
  /// <returns>True if validation conditions passed.</returns>
  public abstract bool IsValidPageDisplayed();
}
```

Figure 5 **The Launch Function**

```
public static T Launch<T>(
  Browser browser = Browser.IE,
  bool clearCookies = true,
  bool maximized = true)
  where T : BasePage, new()
{
  T page = new T();

  var url = page.PageUrl;
  if (url == null)
  {
    throw new InvalidOperationException("Unable to find URL for requested page.");
  }

  var pathToBrowserExe = FacebookCodedUITestProject
        .BrowserFactory.GetBrowserExePath(browser);

  // Setting the currect browser for the test.
  BrowserWindow.CurrentBrowser = GetBrowser(browser);
  var window = BrowserWindow.Launch(page.ConstructUrl());
  page.BrowserWindow = window;

  if (window == null)
  {
    var errorMessage = string.Format(
        "Unable to run browser under the path: {0}", pathToBrowserExe);
          throw new InvalidOperationException(errorMessage);
            }

            page.Body = (window.CurrentDocumentWindow.GetChildren()[0] as
                        UITestControl) as HtmlControl;

  if (clearCookies)
  {
  BrowserWindow.ClearCookies();
  }

  window.Maximized = maximized;

  if (!page.IsValidPageDisplayed())
  {
    throw new InvalidOperationException("Invalid page is displayed.");
  }

  return page;
}
 }
```

These generic methods are much more useful because they're "single purpose" and reduce varying dimensions of expected input to simple types. Under the hood, both functions call the FindAll<T> function and operate on its result, but the implementation is hidden inside their bodies.

Working with Any Browser

I've already put some effort into finding and retrieving controls, but to test whether my functions are implemented correctly I need to get a Web browser working, which means it needs to be launched. Launching a particular browser is as easy as any other browser-related operation. As I mentioned earlier, I want to place all browser-related operations inside page-related classes. However, launching the browser is not a part of testing—it's a prerequisite. Motivated by software development best practices, I decided to create a BasePage class, shown in **Figure 4**, to store common operations for all derived page classes (including launching a browser) without any redundancy.

The static, generic Launch<T> function is also a part of the BasePage class. Inside the function body, a new instance of the specific page type (derived from BasePage) is initialized based on the parameterless default constructor. Later in the code, the target Web browser is set based on the value of the browser parameter ("ie" for Internet Explorer, "chrome" for Google Chrome and so on). This assignment specifies the browser against which the current test will be executed. The next step is to navigate to some URL in the selected browser. This is handled by BrowserWindow.Launch(page.ConstructUrl()), where the ConstructUrl function is a specific function for each derived page. After launching the browser window and navigating to the specific URL, I store the HTML body inside the BasePage property and optionally maximize the browser window (this is desirable because the page controls might overlap and automated UI actions could fail). I then clear the cookies, because each test should be independent. Finally, in the Launch function shown in **Figure 5**, I want to check whether the currently displayed page is the correct one, so I call IsValidPageDisplayed, which will execute in the context of the generic page. This function finds all required HTML controls (login, password, submit button) and validates they exist on the page.

Web browsers evolve continuously and you may not realize when this happens. Sometimes this means certain features aren't available in the new browser version, which in turn causes some tests to fail, even if they were passed previously. That's why it's important to disable automatic browser updates and wait until the new version is supported by the Selenium components for Coded UI Cross-Browser Testing. Otherwise, unexpected exceptions might occur during run time, as shown in **Figure 6**.

Testing, Testing, Testing

Finally, I'll write some logic for my tests. As I mentioned earlier, I want to test two basic user scenarios. The first is a positive login process (a second negative test case is available in the project source code, which you can find at archive.msdn.microsoft.com/mag201312Testing). To get this test running, I must create a specific page class that derives from the BasePage, as shown in **Figure 7**. Inside my new class, in private fields, I place all the constant values (controls, Ids and CSS class names) and create dedicated methods that use those constants to extract specific UI elements from the current page. I also create a function called TypeCredentialAndClickLogin(string login, string password) that fully encapsulates the login operation. At run time, it finds all required controls, simulates the typing in of values passed as parameters, and then presses the Login button by clicking the left mouse button.

After I create the required components, I can build a test case. This test function will validate that the login operation completes successfully. At the beginning of the test case, I launch the Login page using the Launch<T> static function. I pass all required values into the login and password input fields, then click the Login button. When the operation finishes, I validate that the newly displayed panel is a Profile page:

```
[TestMethod]
public void FacebookValidLogin()
{
  var loginPage = BasePage.Launch<LoginPage>();
  loginPage.TypeCredentialAndClickLogin(fbLogin, fbPassword);

  var profilePage = loginPage.InitializePage<ProfilePage>();
  Assert.IsTrue(profilePage.IsValidPageDisplayed(),
    "Profile page is not displayed.");
}
```

While searching for a control with a specific CSS class, I noticed that in the Coded UI Test Framework a complication might arise. In HTML, controls can have more than one class name in the class attribute, and this of course affects the framework I'm working with. If, for example, my current Web site contains a DIV element with attribute class "A B C" and I use the SearchSelector.Class property to find all controls with CSS class "B," I might not get any result—because "A B C" is not equal to "B." To deal with this, I introduce the star "*" notation,

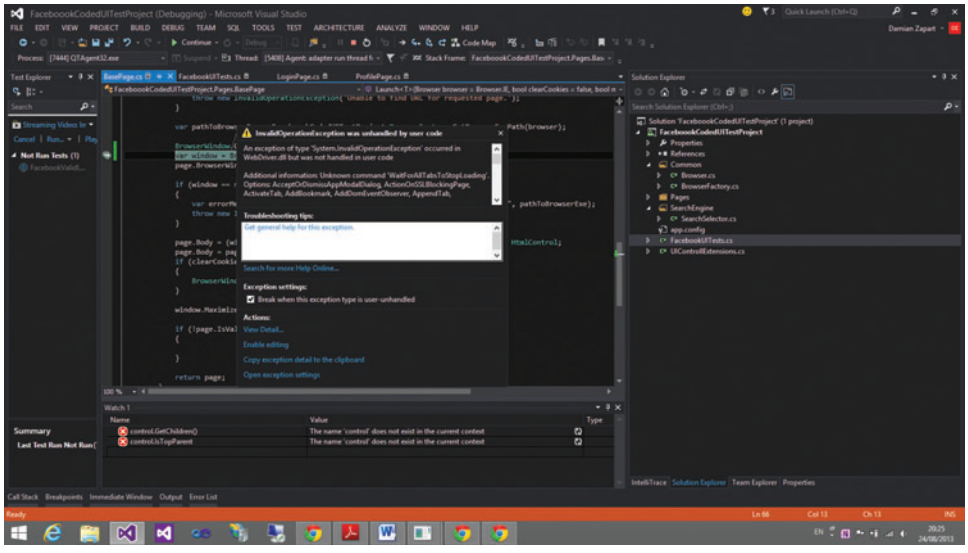


Figure 6 **An Exception After a Web Browser Update**

Figure 7 **The Login Page**

```
public class LoginPage : BasePage
  {
    private const string LoginButtonId = "u_0_1";
    private const string LoginTextBoxId = "email";
    private const string PasswordTextBoxId = "pass";
    private const string LoginFormId = "loginform";
    private const string ErrorMessageDivClass = "login_error_box";
    private const string Page = "login.php";

    /// <summary>
    /// Builds URL for the page.
    /// </summary>
    /// <returns>Uri of the specific page.</returns>
    protected override Uri ConstructUrl()
    {
      this.PageUrl = new Uri(string.Format("{0}/{1}", BasePage.BaseURL,
        LoginPage.Page));
      return this.PageUrl;
    }

    /// <summary>
    /// Validate that the correct page is displayed.
    /// </summary>
    public override bool IsValidPageDisplayed()
    {
      return this.Body.FindById<HtmlDiv>(LoginTextBoxId) != null;
    }

    /// <summary>
    /// Gets the login button from the page.
    /// </summary>
    public HtmlInputButton LoginButton
    {
      get
      {
        return this.Body.FindById<HtmlInputButton>(LoginButtonId);
      }
    }

    /// <summary>
    /// Gets the login textbox from the page.
    /// </summary>
    public HtmlEdit LoginTextBox
    {
      get
      {
        return this.Body.FindById<HtmlEdit>(LoginTextBoxId);
      }
    }

    /// <summary>
    /// Gets the password textbox from the page.
    /// </summary>
    public HtmlEdit PasswordTextBox
    {
      get
      {
        return this.Body.FindById<HtmlEdit>(PasswordTextBoxId);
      }
    }

    /// <summary>
    /// Gets the error dialog window - when login failed.
    /// </summary>
    public HtmlControl ErrorDialog
    {
      get
      {
        return this.Body.FindFirstByCssClass<HtmlControl>("*login_error_box ");
      }
    }

    /// <summary>
    /// Types login and password into input fields and clicks the Login button.
    /// </summary>
    public void TypeCredentialAndClickLogin(string login, string password)
    {
      var loginButton = this.LoginButton;
      var emailInput = this.LoginTextBox;
      var passwordInput = this.PasswordTextBox;

      emailInput.TypeText(login);
      passwordInput.TypeText(password);

      Mouse.Click(loginButton, System.Windows.Forms.MouseButtons.Left);
    }
  }
```

which changes class expectation from "equals" to "contains." So, to get this example working, I need to change class "B" to class "*B."

What If ...

Sometimes tests fail and you have to ask why. In many cases, reviewing the test log is all that's needed to answer this question—but not always. In the Coded UI Test Framework, a new feature provides extra information on demand.

Assume the test failed because a page different than expected was displayed. In the logs, I see that some required control was not found. This is good information, but it doesn't give the full answer. With the new feature, however, I can capture the currently displayed screen. To use that feature, I just have to add the capture and a way to save it in the test cleanup method, as shown in **Figure 8**. Now I get detailed information for any test that fails.

Wrapping Up

In this article I showed how quick and easy it is to start using the new Coded UI Test Framework in Visual Studio 2013 RC. Of course, I described only the basic use of this technology, including managing different browsers, and supporting a variety of operations to find, retrieve and manipulate HTML controls. There are many more great features worth exploring. ■

Figure 8 **The Test Cleanup Method**

```
[TestCleanup]
public void TestCleanup()
{
  if (this.TestContext.CurrentTestOutcome != null &&
    this.TestContext.CurrentTestOutcome.ToString() == "Failed")
{
    try
    {
      var img = BrowserWindow.Desktop.CaptureImage();
      var pathToSave = System.IO.Path.Combine(
        this.TestContext.TestResultsDirectory,
        string.Format("{0}.jpg", this.TestContext.TestName));
      var bitmap = new Bitmap(img);
      bitmap.Save(pathToSave);
    }
    catch
    {
      this.TestContext.WriteLine("Unable to capture or save screen.");
    }
  }
}
```

Damian Zapart *is a .NET developer with more than seven years of experience. He focuses mainly on Web-based technologies. He's a programming geek interested in cutting-edge technologies, design patterns and testing. Visit his blog at bit.ly/18BV7Qx to read more about him.*

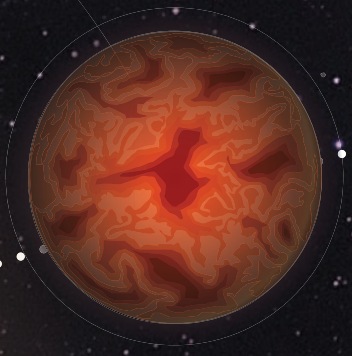
Thanks to the following Microsoft technical experts for reviewing this article: *Bilal A. Durrani and Evren Önem*

Visual Studio® LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

2014 Dates Announced

Much like outer space, the .NET development platform is ever-changing and constantly evolving. Visual Studio Live! exists to help guide you through this universe, featuring code-filled days, networking nights and independent education. Whether you are a .NET developer, software architect or a designer, Visual Studio Live!'s multi-track events include focused, cutting-edge education on the .NET platform.



LAS VEGAS

LIVE! 360 DEV

PLANET HOLLYWOOD RESORT AND CASINO

COMPREHENSIVE TRAINING FOR THE DEVELOPER WORLD

LIVE! Visual Studio | LIVE! SQL Server | LIVE! SharePoint | LIVE! Web Dev | LIVE! Modern Apps

Visual Studio Live! Las Vegas

Part of Live! 360 Dev

March 10 – 14
Planet Hollywood
Las Vegas, NV

The Developer World is always expanding; new technologies emerge, current ones evolve and demands on your time grow. Live! 360 Dev offers comprehensive training on the most relevant technologies in your world today.

Visual Studio Live! Chicago

May 5 – 8
Chicago Hilton
Chicago, IL

YOUR GUIDE TO THE .NET DEVELOPMENT UNIVERSE

ORLANDO

LIVE! 360

LOEWS ROYAL PACIFIC RESORT

IT AND DEVELOPER TRAINING THAT'S OUT OF THIS WORLD

LIVE! Visual Studio | LIVE! SharePoint | LIVE! Web Dev | LIVE! Modern Apps

Visual Studio Live! Redmond

August 18 – 22
Microsoft Conference Center
Redmond, WA

Visual Studio Live! Orlando

Part of Live! 360

November 17 – 21
Loews Royal Pacific Resort, Orlando, FL

Live! 360 is a unique conference, created for the IT and Developer worlds to come together to explore leading edge technologies and conquer current ones.

LIVE!

WHERE WILL YOU LAUNCH YOUR TRAINING?

vslive.com
live360events.com

CONNECT WITH VISUAL STUDIO LIVE!

- twitter: @VSLive
- facebook.com/VSLiveEvents
- linkedin.com – Join the Visual Studio Live group

An Introduction to Model-Based Testing and Spec Explorer

Yiming Cao and Sergio Mera

Producing high-quality software demands a significant effort in testing, which is probably one of the most expensive and intensive parts of the software development process. There have been many approaches for improving testing reliability and effectiveness, from the simplest functional black-box tests to heavyweight methods involving theorem provers and formal requirement specifications. Nevertheless, testing doesn't always include the necessary level of thoroughness, and discipline and methodology are often absent.

Microsoft has been successfully applying model-based testing (MBT) to its internal development process for more than a decade now. MBT has proven a successful technique for a variety of internal and external software products. Its adoption has steadily increased over the years. Relatively speaking, it has been well received in the testing community, particularly when compared with other methodologies living on the "formal" side of the testing spectrum.

Spec Explorer is a Microsoft MBT tool that extends Visual Studio, providing a highly integrated development environment for creating behavioral models, plus a graphical analysis tool for checking the validity of those models and generating test cases from them.

This article discusses:

- What model-based testing is all about
- How to create models using Spec Explorer with Visual Studio
- Using a chat system as an example to explore model-based testing
- Advantages and disadvantages of model-based testing

Technologies discussed:

Visual Studio, Spec Explorer

We believe this tool was the tipping point that facilitated the application of MBT as an effective technique in the IT industry, mitigating the natural learning curve and providing a state-of-the-art authoring environment.

In this article we provide a general overview of the main concepts behind MBT and Spec Explorer, presenting Spec Explorer via a case study to showcase its main features. We also want this article to serve as a collection of practical rules of thumb for understanding when to consider MBT as a quality assurance methodology for a specific testing problem. You shouldn't blindly use MBT in every testing scenario. Many times another technique (like traditional testing) might be better a choice.

What Makes a Testable Model in Spec Explorer?

Even though different MBT tools offer different functionality and sometimes have slight conceptual discrepancies, there's general agreement about the meaning of "doing MBT." Model-based testing is about automatically generating test procedures from models.

Models are usually manually authored and include system requirements and expected behavior. In the case of Spec Explorer, test cases are automatically generated from a state-oriented model. They include both test sequences and the test oracle. Test sequences, inferred from the model, are responsible for driving the system under test (SUT) to reach different states. The test oracle tracks the evolution of the SUT and determines if it conforms to the behavior specified by the model, emitting a verdict.

The model is one of the main pieces in a Spec Explorer project. It's specified in a construct called model programs. You can write model programs in any .NET language (such as C#). They consist of a set of rules that interact with a defined state. Model programs

Figure 1 **Action Declarations**

```
// Cord code
config ChatConfig
{
  action void LogonRequest(int user);
  action event void LogonResponse(int user);

  action void LogoffRequest(int user);
  action event void LogoffResponse(int user);

  action void ListRequest(int user);
  action event void ListResponse(int user, Set<int> userList);

  action void BroadcastRequest(int senderUser, string message);
  action void BroadcastAck(int receiverUser, int senderUser, string message);

  // ...
}
```

are combined with a scripting language called Cord, the second key piece in a Spec Explorer project. This permits specifying behavioral descriptions that configure how the model is explored and tested. The combination of the model program and the Cord script creates a testable model for the SUT.

Of course, the third important piece in the Spec Explorer project is the SUT. It isn't mandatory to provide this to Spec Explorer to generate the test code (which is the Spec Explorer default mode), because the generated code is inferred directly from the testable model, without any interaction with the SUT. You can execute test cases "offline,"

Figure 2 **The Model's State**

```
/// <summary>
/// A model of the MS-CHAT sample.
/// </summary>
public static class Model
{
  /// <summary>
  /// State of the user.
  /// </summary>
  enum UserState
  {
    WaitingForLogon,
    LoggedOn,
    WaitingForList,
    WatingForLogoff,
  }

  /// <summary>
  /// A class representing a user
  /// </summary>
  partial class User
  {
    /// <summary>
    /// The state in which the user currently is.
    /// </summary>
    internal UserState state;

    /// <summary>
    /// The broadcast messages that are waiting for delivery to this user.
    /// This is a map indexed by the user who broadcasted the message,
    /// mapping into a sequence of broadcast messages from this same user.
    /// </summary>
    internal MapContainer<int, Sequence<string>> waitingForDelivery =
      new MapContainer<int,Sequence<string>>();

  }

    /// <summary>
    /// A mapping from logged-on users to their associated data.
    /// </summary>
    static MapContainer<int, User> users = new MapContainer<int,User>();
      // ...
  }
```

decoupled from model evaluation and test-case generation stages. However, if the SUT is provided, Spec Explorer can validate that the bindings from the model to the implementation are well-defined.

Case Study: A Chat System

Let's take a look at one example to show how you can build a testable model in Spec Explorer. The SUT in this case is going to be a simple chat system with a single chat room where users can log on and log off. When a user is logged on, he can request the list of the logged-on users and send broadcast messages to all users. The chat server always acknowledges the requests. Requests and responses behave asynchronously, meaning they can be intermingled. As expected in a chat system, though, multiple messages sent from one user are received in order.

One of the advantages of using MBT is that, by enforcing the need to formalize the behavioral model, you can get a lot of feedback for the requirements. Ambiguity, contradictions and lack of context can surface at early stages. So it's important to be precise and formalize the system requirements, like so:

R1. Users must receive a response for a logon request.
R2. Users must receive a response for a logoff request.
R3. Users must receive a response for a list request.
R4. List response must contain the list of logged-on users.
R5. All logged-on users must receive a broadcast message.
R6. Messages from one sender must be received in order.

Spec Explorer projects use actions to describe interaction with the SUT from the test-system standpoint. These actions can be call actions, representing a stimulus from the test system to the SUT; return actions, capturing the response from the SUT (if any); and event actions, representing autonomous messages sent from the SUT. Call/return actions are blocking operations, so they're represented by a single method in the SUT. These are the default action declarations, whereas the "event" keyword is used to declare an event action. **Figure 1** shows what this looks like in the chat system.

With the actions declared, the next step is to define the system behavior. For this example, the model is described using C#. System state is modeled with class fields, and state transitions are modeled with rule methods. Rule methods determine the steps you can take from the current state in the model program, and how state is updated for each step.

Because this chat system essentially consists of the interaction between users and the system, the model's state is just a collection of users with their states (see **Figure 2**).

As you can see, defining a model's state isn't much different from defining a normal C# class. Rule methods are C# methods for describing in what state an action can be activated. When that happens, it also describes what kind of update is applied to the model's state. Here, a "LogonRequest" serves as an example to illustrate how to write a rule method:

```
[Rule]
static void LogonRequest(int userId)
{
  Condition.IsTrue(!users.ContainsKey(userId));
  User user = new User();
  user.state = UserState.WaitingForLogon;
  user.waitingForDelivery = new MapContainer<int, Sequence<string>>();
  users[userId] = user;
}
```

This method describes the activation condition and update rule for the action "LogonRequest," which was previously declared in Cord code. This rule essentially says:

- The LogonRequest action can be performed when the input userId doesn't yet exist in the current user set. "Condition.IsTrue" is an API provided by Spec Explorer for specifying an enabling condition.
- When this condition is met, a new user object is created with its state properly initialized. It's then added to the global users collection. This is the "update" part of the rule.

At this point, the majority of the modeling work is finished. Now let's define some "machines" so we can explore the system's behavior and get some visualization. In Spec Explorer, machines are units of exploration. A machine has a name and an associated behavior defined in the Cord language. You can also compose one machine with others to form more complex behavior. Let's look at a few example machines for the chat model:

```
machine ModelProgram() : Actions
{
  construct model program from Actions where scope = "Chat.Model"
}
```

The first machine we define is a so-called "model program" machine. It uses the "construct model program" directive to tell Spec Explorer to explore the entire behavior of the model based on rule methods found in the Chat.Model namespace:

```
machine BroadcastOrderedScenario() : Actions
{
  (LogonRequest({1..2}); LogonResponse){2};
  BroadcastRequest(1, "1a");
  BroadcastRequest(1, "1b");
  (BroadcastAck)*
}
```

The second machine is a "scenario," a pattern of actions defined in a regular-expression-like way. Scenarios are usually composed with a "model program" machine in order to slice the full behavior, as in the following:

```
machine BroadcastOrderedSlice() : Actions
{
  BroadcastOrderedScenario || ModelProgram
}
```

The "||" operator creates a "synchronized parallel composition" between the two participating machines. The resulting behavior will contain only the steps that can be synchronized on both machines (by "synchronized" we mean have the same action with the same argument list). Exploring this machine results in the graph shown in **Figure 3**.

As you can see from the graph in **Figure 3**, the composed behavior complies with both the scenario machine and the model program. This is a powerful technique for getting a simpler subset of a complex behavior. Also, when your system has infinite state space (as in the case of the chat system), slicing the full behavior can generate a finite subset more suitable for testing purposes.

Let's analyze the different entities in this graph. Circle states are controllable states. They're states where stimuli are provided to the SUT. Diamond states are observable states. They're states where one or more events are expected from the SUT. The test oracle (the expected result of testing) is already encoded in the graph with event steps and their arguments. States with multiple outgoing event steps are called non-deterministic states, because the event the SUT provides at execution time isn't determined at modeling time. Observe that the exploration graph in **Figure 3** contains several non-deterministic states: S19, S20, S22 and so forth.

The explored graph is useful for understanding the system, but it's not yet suitable for testing because it isn't in "test normal" form. We say a behavior is in test normal form if it doesn't contain any state that has more than one outgoing call-return step. In the graph in **Figure 3**, you can see that S0 obviously violates this rule. To convert such behavior into test normal form, you can simply create a new machine using the test cases construction:

```
machine TestSuite() : Actions where TestEnabled = true
{
  construct test cases where AllowUndeterminedCoverage = true
  for BroadcastOrderedSlice
}
```

This construct generates a new behavior by traversing the original behavior and generating traces in test normal form. The traversal criterion is edge coverage. Each step in the original behavior is covered at least once. The graph in **Figure 4** shows the behavior after such traversal.

To achieve test normal form, states with multiple call-return steps are split into one per step. Event steps are never split and are always fully preserved, because events are the choices that the SUT can make at runtime. Test cases must be prepared to deal with any possible choice.

Spec Explorer can generate test suite code from a test normal form behavior. The default form of generated test code is a Visual Studio unit test. You can directly execute such a test suite with the Visual Studio test tools, or with the mstest.exe command-line tool. The generated test code is human readable and can be easily debugged:

```
#region Test Starting in S0
[Microsoft.VisualStudio.TestTools.UnitTesting.TestMethodAttribute()]
public void TestSuiteS0() {
  this.Manager.BeginTest("TestSuiteS0");
  this.Manager.Comment("reaching state \'S0\'");
  this.Manager.Comment("executing step \'call LogonRequest(2)\'");
  Chat.Adapter.ChatAdapter.LogonRequest(2);
  this.Manager.Comment("reaching state \'S1\'");
  this.Manager.Comment("checking step \'return LogonRequest\'");
  this.Manager.Comment("reaching state \'S4\'");
  // ...
    }
```

The test-code generator is highly customizable and can be configured to generate test cases that target different test frameworks, such as NUnit.

The full Chat model is included in the Spec Explorer installer.

When Does MBT Pay Off?

There are pros and cons when using model-based testing. The most obvious advantage is that after the testable model is completed, you can generate test cases by pushing a button. Moreover, the fact that a model has to be formalized up front enables early detection of requirement inconsistencies and helps teams to be in accord in terms of expected behavior. Note that when writing manual test cases, the "model" is also there, but it's not formalized and lives in the head of the tester. MBT forces the test team to clearly communicate its expectations in terms of system behavior and write them down using a well-defined structure.

Another clear advantage is that project maintenance is lower. Changes in system behavior or newly added features can be reflected by updating the model, which is usually much simpler than changing manual test cases, one by one. Identifying only the

NEVRON
visualize your success

www.nevron.com
email@nevron.com
+1-888-201-6088
(Toll free, USA and Canada)

Nevron Data Visualization

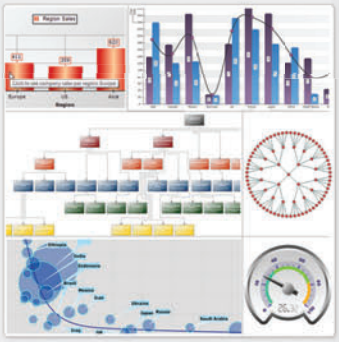
The leading data visualization components for a wide range of .NET platforms.14+ years of refinement, complete feature sets, highly customizable design and great support.

Developers

NEVRON Open VISION

Nevron Open Visions is a Cross - Platform Presentation Layer based on .NET. It aims to run on all major operating systems and integrate with already existing .NET based presentation layers.

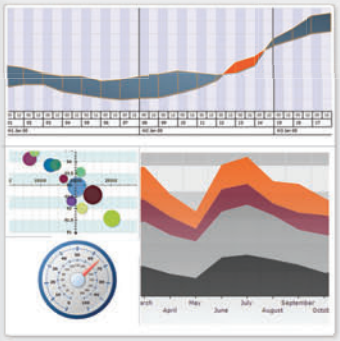
NEVRON VISION for .NET



Nevron Vision for .NET includes chart, gauge, diagram, map and user interfaces components that help you create enterprise-grade digital dashboards, scorecards, diagrams, maps, MMI interfaces and much more.

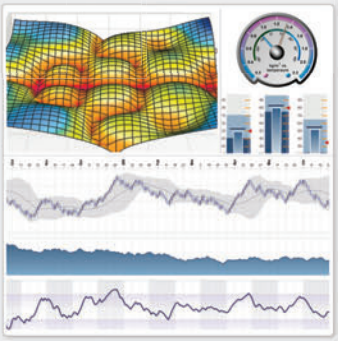
IT Professionals

NEVRON VISION for SharePoint



Nevron Vision for SharePoint combines the leading Chart and Gauge web parts for SharePoint 2007, 2010 and 2013. With this suite you can easily convert your SharePoint pages into interactive dashboards and reports.

NEVRON VISION for SSRS



Nevron Vision for SSRS presents the leading data visualization report items for SSRS 2005, 2008 and 2012. The Chart and Gauge help you deliver deep data insights with highly customizable engaging looks.

Nevron components integrate seamlessly in Web and Desktop .NET applications, SQL Server Reporting Services reports and SharePoint portals. All data visualization tools deliver an unmatched set of enterprise- grade features which makes Nevron the trusted vendor for many Fortune 500 companies.

Download you free evaluation copy from **www.nevron.com** today.

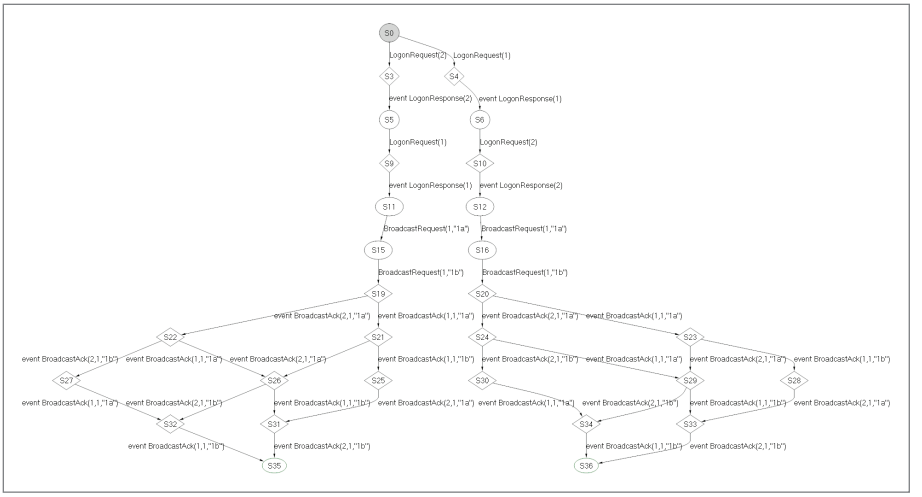


Figure 3 **Composing Two Machines**

test cases that need to be changed is sometimes a time-consuming task. Consider as well that model authoring is independent of the implementation or the actual testing. That means that different members of a team can work on different tasks concurrently.

On the downside, a mindset adjustment is often required. This is probably one of the major challenges of this technique. On top of the well-known problem that people in the IT industry don't have time to try out new tools, the learning curve for using this technique isn't negligible. Depending on the team, applying MBT might require some process changes as well, which can also generate some push back.

The other disadvantage is that you have to do more work in advance, so it takes more time to see the first test case being generated, compared with using traditional, manually written test cases. Additionally, the complexity of the testing project needs to be great enough to justify the investment.

Luckily, there are some rules of thumb we believe help identify when MBT really pays off. Having an infinite set of system states with requirements you can cover in different ways is a first sign. A reactive or distributed system, or a system with asynchronous or non-deterministic interactions is another. Also, methods that have many complex parameters can point in the MBT direction.

When these conditions are met, MBT can make a big difference and save significant testing effort. An example of this is Microsoft Blueline, a project where hundreds of protocols were verified as part of the Windows protocol compliance initiative. In this project, we used Spec Explorer to verify the technical accuracy of protocol documentation with respect to the actual protocol behavior. This was a gigantic effort and Microsoft spent around 250 person-years in testing. Microsoft Research validated a statistical study that showed using MBT saved Microsoft 50 person-years of tester work, or around 40 percent of the effort compared with a traditional testing approach.

Model-based testing is a powerful technique that adds a systematic methodology to traditional techniques. Spec Explorer is a mature tool that leverages the MBT concepts in a highly integrated, state-of-the-art development environment as a free Visual Studio Power Tool. ■

Yiming Cao *is a senior development lead for the Microsoft Interop and Tools team and works on the Protocol Engineering Framework (including Microsoft Message Analyzer) and Spec Explorer. Before joining Microsoft he worked for IBM Corp. on its enterprise collaboration suite, and then joined a startup company working on media-streaming technologies.*

Sergio Mera *is a senior program manager for the Microsoft Interop and Tools team and works on the Protocol Engineering Framework (including Microsoft Message Analyzer) and Spec Explorer. Before joining Microsoft he was a researcher and lecturer for the Computer Science Department at the University of Buenos Aires and worked on modal logics and automated theorem proving.*

Thanks to the following technical expert for reviewing this article: *Nico Kicillof (Microsoft)*

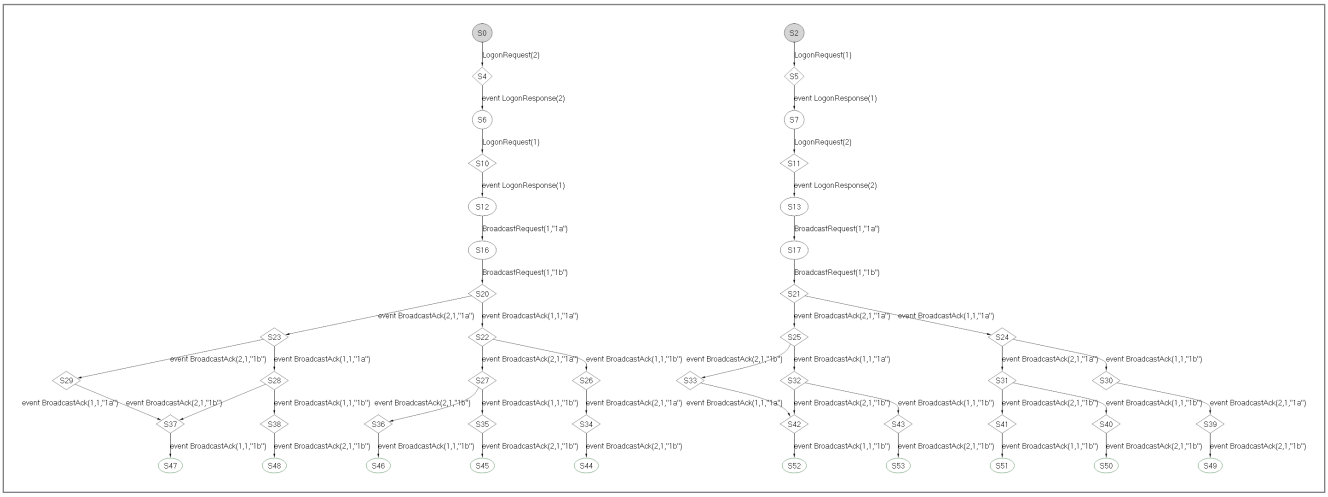


Figure 4 **Generating New Behavior**

ALEXSYS TEAM®

WORKFLOW APPLICATIONS | HELP DESK | BUG TRACKING

MADE EASY!

Alexsys Team® offers ***flexible task management*** software for Windows, Web, and Mobile Devices. Track whatever you need to get the job done - anywhere, anytime on practically any device!

Thousands are using Alexsys Team® to create workflow solutions and web apps - without coding!
Fortune 500 companies, state, local, DOD, and other federal agencies use Team to manage their tasks. Easily tailor Team to meet your exact requirements - even if they change daily, weekly, or even every minute.

Alexsys Team® Features Include:

- Form and Database customization
- Custom workflows
- Role-based security
- Adaptive Rules Engine
- DOD CAC card support
- Time recording
- Automated Escalations, Notifications, and SOAP Messaging
- Supports MS-SQL, MySQL, and Orcale databases

Our renowned tech support team is here to make you and your team a success. Don't have enough resources? Our professional services staff has helped companies large and small use Alexsys Team® at a fraction of the cost of those big consulting firms.

Find out yourself:
Free Trial and Single User FreePack™ available at Alexcorp.com

FIND OUT MORE!

1-888-880-ALEX (2539)
ALEXCORP.COM

This article is from *MSDN Magazine*'s special coverage of application development in the government sector.
Read more articles like this in the Special Government Issue (msdn.microsoft.com/magazine/dn463800) of *MSDN Magazine*.

Freedom of Information Act Data at Your Fingertips

Vishwas Lele

The Freedom of Information Act (FOIA) gives U.S. citizens access to government data to help educate themselves and make more informed decisions. The process of getting this data, however, can be complex and daunting. In this article, I'll walk through the building of a Windows Store app, called MyFOIA, that lets users more easily get this information, share it with others and even get automatic updates on FOIA requests.

What Is the FOIA?

The FOIA grants citizens the right to request information from any government agency. The agency is then required to provide the citizen with any pertinent, non-classified information. This law is designed to increase governmental transparency. However, the request process doesn't allow for easy access to an agency's information.

Most agencies offer frequently requested information on their Web sites. For information that doesn't appear there, a person must make an FOIA request. This process often requires the citizen to submit a written request by postal mail or e-mail, outlining in as much detail as possible the information the person wants and the desired format in which it should be provided. The request must be compliant with certain specific format guidelines, which vary from agency to agency. The typical return time on an FOIA request can be at least one month. These factors raise the need for a tool that can improve access to this type of government information.

In a system of government designed by the people, for the people, the FOIA helps citizens access information that previously would've been available only to federal agencies.

The law has helped citizens uncover results from unpublished studies, previously withheld videos and unreported breaches of regulations, all of which wouldn't have been otherwise available to the public. See "The FOIA in Action" in this article for details about some real examples.

Challenges of the FOIA

The FOIA can be a powerful tool for analyzing and extracting useful conclusions from government data. However, the process is long and complicated. Under the current system, the citizen must first figure out—as specifically as possible—what he's looking for. Sometimes the data a citizen wants is locked in an agency-specific portal.

To combat this problem, there have been some multi-agency databases created, such as FOIAonline (1.usa.gov/OOs4zu). Here, the citizen has two options: search the frequently requested FOIA data or make an official FOIA request. The official FOIA request requires divulging personal information to create an account, in addition to submitting a detailed letter of description outlining the request.

Depending on the agency, the requester may also be required to submit a fee. This process can be arduous and relies on the citizen actively seeking the information from the agency by means of a request, rather than having the data widely available at the person's fingertips. A user can't receive notification upon any updates to the data in the system, and there's no easy way for people to share or discuss this information by means of social media.

Code download available at github.com/AppliedIS/Foia.

Enter the MyFOIA App

I decided it would be useful to build a Windows Store app to make FOIA-related information more accessible. Users would be able to easily access FOIA data by searching for existing FOIA requests and creating new ones. Furthermore, a user would be notified of any updates to an FOIA request she wanted to follow. Finally, the app would make it easy to share interesting tidbits related to FOIA data over social media, thereby encouraging others to get involved.

I'll now briefly review the functionality provided by the app.

Home Page **Figure 1** depicts the Home page of the MyFOIA app. Users can search for requests, make a new request (as a guest) and generate reports. The app bar also allows for navigation controls and enables users to tweet selected text from within the app.

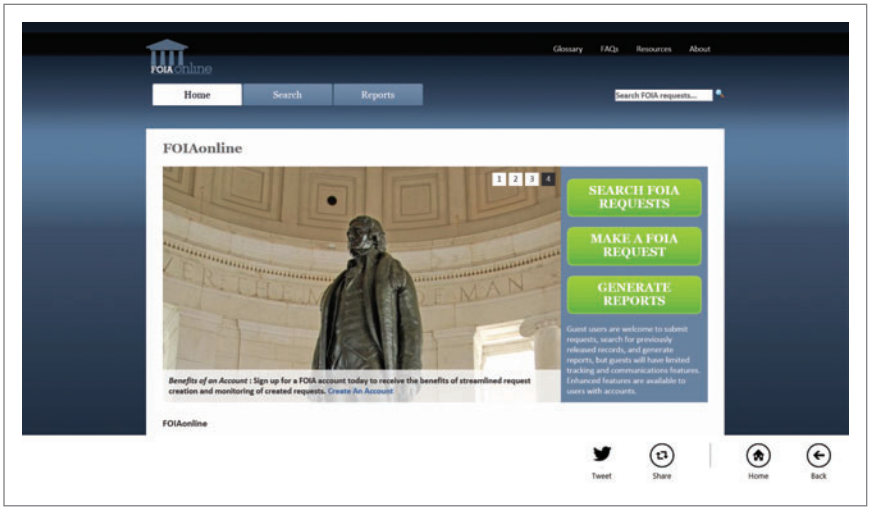


Figure 1 **The MyFOIA App Home Page**

Search Page **Figure 2** depicts the Search page of the MyFOIA app. Users can search for requests, appeals and records released in response to FOIA requests. In this example, the user is searching through the U.S. Environmental Protection Agency's (EPA) FOIA records for the search term "CO2."

Search Results **Figure 3** depicts the search results page. Results of the search placed earlier (in **Figure 2**) are displayed on the screen. Notice the column at the right with the heading of Notify. Clicking on one or more checkboxes allows users to receive notifications if there's an update related to the request.

Select and Share Page **Figure 4** depicts the "select and share" page that lets users select interesting tidbits from responses and share them over Twitter. Notice the browser window with the Twitter page alongside the MyFOIA app.

High-Level Architecture

Figure 5 depicts the high-level architecture of the MyFOIA app, which was built using JavaScript and HTML. It relies on the Windows 8.1 WebView control to host content from the FOIAonline site, thereby making available the core FOIAonline functionality (create, search and report on responses). The app also extends functionality provided by the FOIAonline Web site by "injecting" JavaScript into the WebView-hosted content. For example, it lets users track FOIA responses. If there's an update to an FOIA response, all devices that have registered to receive updates will receive push notifications. The MyFOIA app also extends the look and feel of the FOIAonline Web site to match a native Windows Store app. This includes enabling "pinch and zoom," relying on an app bar for navigation, applying necessary styling to make the app touch-friendly and sharing text with other Windows Store apps.

Debrief: The Freedom of Information Act (FOIA)

Transparency of government data plays a vital role in keeping U.S. citizens informed of the activities of government agencies and involved in the governing process. The FOIA makes this data available, but requesting and receiving this information can be a daunting and cumbersome experience. This article details the development of a Windows Store app that improves on the current processes for working with FOIA requests.

IT Brief:

This article can serve as an example for government agencies of how to improve the process of fulfilling the FOIA mandate. As an alternative to developing completely new solutions, it shows how emerging technologies can augment and improve current systems. Important considerations include the following:

- Government agencies are required to comply with FOIA requests.
- Current processes for fulfilling requests vary from agency to agency.
- Filing and tracking FOIA requests can be difficult for citizens, but the process can be standardized and improved.
- Using new technology such as Windows Store apps can help standardize and enhance the process.

Dev Brief:

Developers might not always have the time or resources to completely rebuild current systems. This article shows how to use modern technologies, such as the new and improved WebView control in Windows 8.1, to interact with and improve existing systems. In this case, an app works with an existing FOIA-related Web site and adds the following capabilities:

- Users can be notified if there's a change to an FOIA request in which they were interested and following.
- Users can easily share FOIA-related data over social media.
- Users can interact with the app smoothly via a variety of input sources, including touch, pen, mouse and keyboard.

More Information:

- FOIAonline Web site: 1.usa.gov/00s4zu
- Windows 8.1 WebView control sample: bit.ly/19zxAPb
- Windows Azure Mobile Services: bit.ly/0p5Vdi
- FOIA in the News - 2004-2006 (The National Security Archive): bit.ly/1ad5Tfd
- Freedom of Information Act Web site: foia.gov

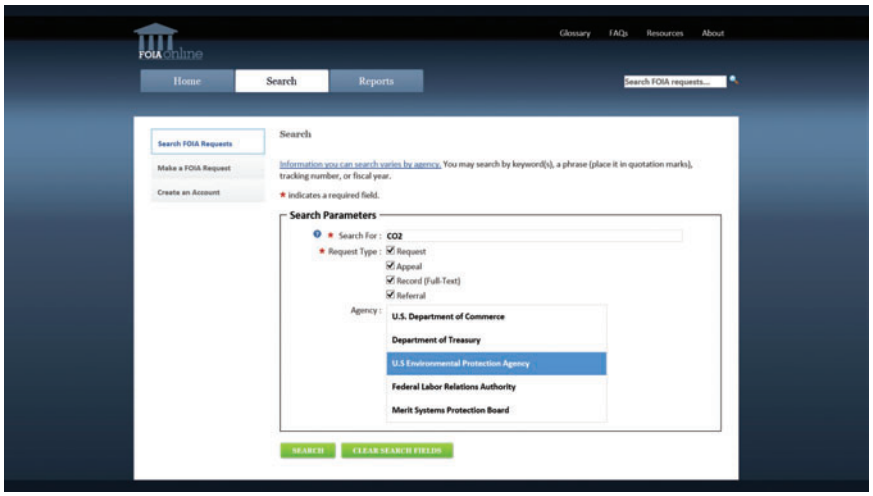


Figure 2 **The MyFOIA App Search Page**

The other key piece of functionality offered by the MyFOIA app is the notification capability. Users can register to receive notifications whenever a given FOIA response is updated. This notification capability is built using Windows Azure Mobile Services (WAMS), which enables back-end capabilities for apps, including:

- Simple provisioning and management of tables where apps can store data.
- Integration with the notification hub to deliver push notifications.
- Client libraries for various devices, including JavaScript libraries.
- The ability to add script-based server logic, including scheduler capabilities.

Each device, at app activation time, requests a push notification channel from the Notification Client Platform. In turn, the Notification Client Platform asks Windows Push Notification Services (WNS) to create a notification channel and return it to the device. Subsequently, when the user enables tracking on a given FOIA response, the MyFOIA app registers the device with the Windows Azure notification hub. In the background, the WAMS scheduler periodically looks for updates to all registered FOIA responses. If the scheduler discovers an update, a push notification is sent to all devices via the notification hub.

Implementation Details

The first challenge in developing the app was that the FOIAonline Web site doesn't currently support an API for app developers. Fortunately, Microsoft has released an updated version of the WebView control as part of Windows 8.1 that lets me host an FOIAonline Web page within a Windows Store app built with native HTML and JavaScript (the earlier version of the WebView control in Windows 8 was accessible only from apps built with XAML and C#).

You're probably thinking this uses some sort of an iframe-based approach. Not quite. The WebView control offers several advantages over an iframe-based approach. Also, some sites simply disallow their content to be loaded within an iframe.

In simple terms, the new WebView control allows an HTML app to host HTML content. Consider the following HTML and JavaScript snippets used to create the WebView control:

```
<x-ms-webview id="foiaWebview"></x-ms-webview>
var foiaWebview = document.createElement("x-ms-webview");
```

Note that x-ms-webview follows vendor-specific extension syntax, because WebView is a Microsoft-specific control. Once you've created the WebView control, you can load the HTML content as a new document using the navigateToString method:

```
x-ms-webview.navigateToString(stringHTMLContent);
```

So far the functionality looks similar to that provided by an iframe, but there are significant differences:

- The WebView control is integrated into the display tree along with other controls on the HTML page, so you can apply styling just as with other controls on the page. This means you can also overlay other controls on top of the WebView control.
- The WebView control offers a set of navigation events that provides the app with an insight into the loading of content. Apps can register for events such as MSWebViewNavigationStarting, MSWebViewContentLoading, MSWebViewContentLoaded and MSWebViewNavigationCompleted, and take appropriate action. In addition, the WebView control also periodically (every 500 ms) fires a LongRunningScriptDetected event that lets the app halt a potentially errant script.

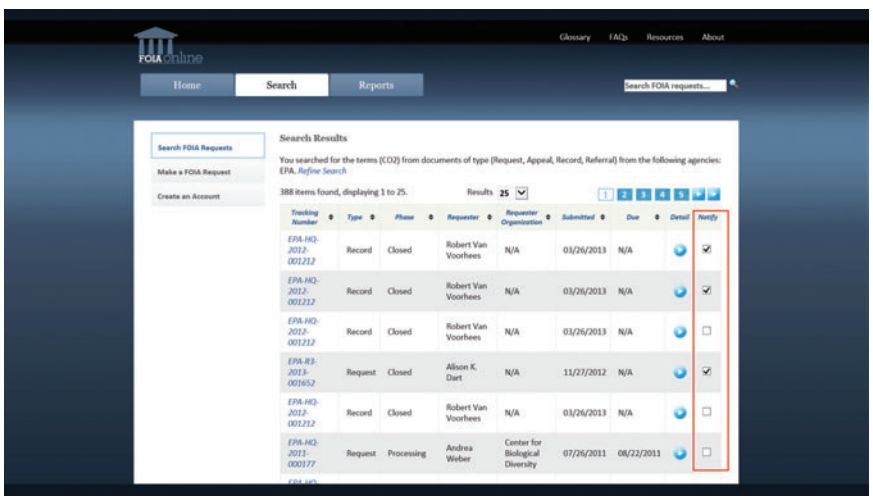


Figure 3 **The MyFOIA App Search Results Page**

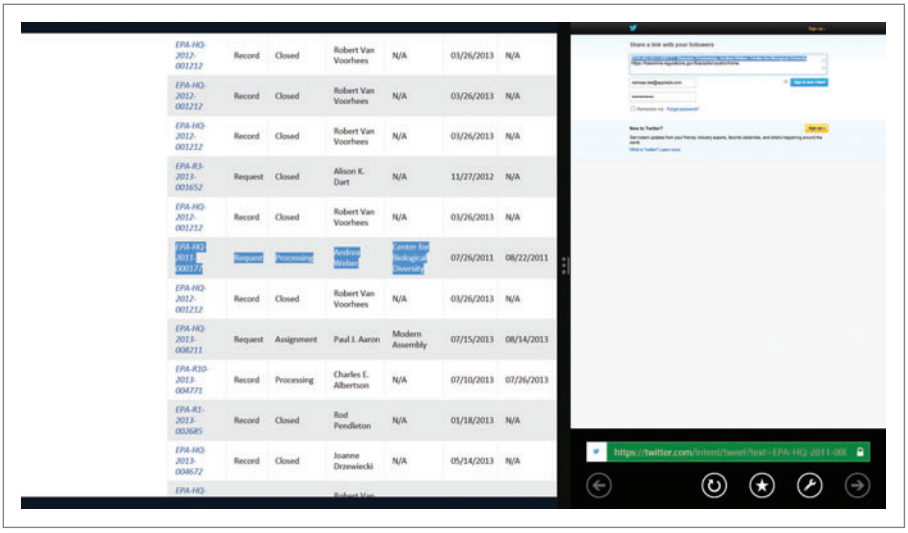


Figure 4 **The MyFOIA App Select and Share Page**

- The WebView control can take advantage of Internet Explorer SmartScreen filtering to block "phishing" attacks. The app can be notified of malicious content and navigate away from the page.
- The WebView control comes with built-in functionality that's commonly needed by apps hosting Web content. This includes checking for the existence of "next" and "previous" links and navigating accordingly, capturing screenshots of the Web content being displayed, and remembering a user's selection.
- For functionality that isn't built-in, the WebView control provides a generic scheme for the app to communicate with the WebView control via the InvokeScriptAsync method and ScriptNotify event. As the name suggests, InvokeScriptNotify provides—as an asynchronous action—the ability to execute a script from within the currently loaded HTML inside the WebView control. Analogously, the HTML page within a WebView can raise a ScriptNotify event within the app. As you can imagine, a number of security checks are imposed in order to prevent externally hosted, malicious Web content from hijacking the app. For example, only the Web sites registered in the app package manifest are allowed to raise events within an app.
- Functionality such as pinch and zoom and phone-number detection works out of the box with the WebView control. Although you can certainly achieve similar behavior using iframes, it would require a bunch of custom JavaScript and CSS.

Now that you're armed with knowledge of how the WebView control works, I'll discuss how the MyFOIA app uses the WebView control. The start page of the MyFOIA app is default.html, as defined in the app package manifest. The default.html page simply hosts the WebView control, along with the WinJS.UI.AppBar control. When the app is activated, the app.onactivated event (defined in default.js) gets invoked. Inside its event handler, you register for the events discussed previously, including MSWebViewNavigationStarting, MSWebViewDOMContentLoaded and MSWebViewScriptNotify. Finally, you navigate to the foiaHome.html page. This is a page bundled with the app package, which is why you'll need an ms-appx-web prefixed URL to access it, like this:

```
var homePageUri = "ms-appx-web:///foiaHome.html";
```

If you look at the contents of this page, it looks similar to the homepage of the FOIAonline site. I bundled this page as part of the app for two reasons. First, I needed a fast, responsive, touch-friendly home screen that stretched to fit the entire on-screen real estate. Second, I wanted to limit the app to the functionality available to guest users only. Bundling a copy of the homepage allowed me to remove the functionality related to registered users. Specifically, the MyFOIA app has no functionality to solicit user credentials.

Figure 6 depicts the homepage of the FOIAonline site. Compare and contrast it to the MyFOIA app Home page (**Figure 1**) and you'll notice that several items (annotated in red on **Figure 6**) have been removed in order to provide a more fluid UI experience.

The next challenge I ran into was enabling some sort of push notification capability that notifies users of updates to an FOIA response in which they were interested. However, no such push notification capability exists within the FOIAonline Web site. Fortunately, the InvokeScriptAsync and ScriptNotify methods discussed earlier let me "inject" the push notification capability within the existing Web page. Refer to **Figure 3** and you'll notice the additional Notify column. This column doesn't exist on the FOIAonline Web site.

I'll explain how I was able to add this additional column. Once DOM content is loaded inside the WebView control, it will fire the foiaWebview_onDOMContentLoaded event (recall that I registered for it inside the app-activated event). This lets the MyFOIA app modify the behavior of the hosted content by injecting a custom script that adds the Notify column to the existing HTML. This code creates a dependence on the underlying Web site. As a result, the MyFOIA app can break if the underlying Web site is modified. In the absence of an API, the WebView control offers the only option to extend the functionality offered by the Web site.

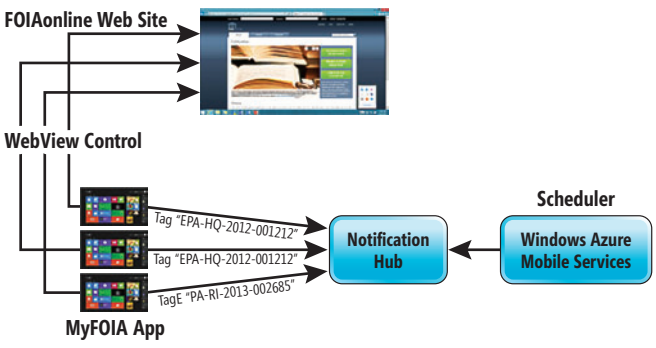


Figure 5 **The High-Level Architecture of the MyFOIA App**

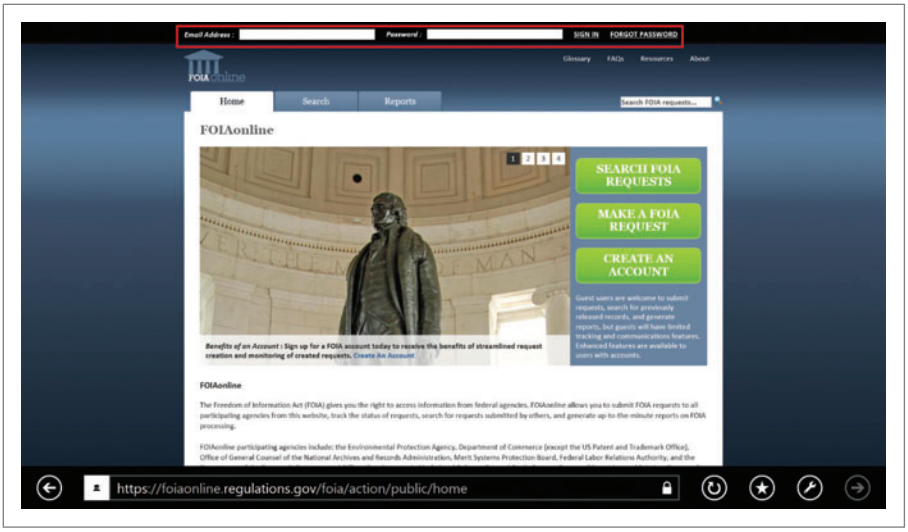


Figure 6 **The FOIAonline Home Page**

In **Figure 7**, you can see that along with adding the Notify column, I pass in the "eval" script function, along with a script argument, to the webviewControl.invokeScriptAsync method.

Upon successfully completing this, the custom script is executed within the loaded HTML page. Note that within the script snippet, passed in as an argument, I include a call to method window.external.notify. So when the user selects one of the checkboxes, an foiaWebview_onScriptNotify event is fired within the MyFOIA app. **Figure 8** depicts how this event is handled by the app. I invoke the WAMS method called getTable to obtain a reference to a specific table. Using the table object, I add a new record containing the number of the FOIA request being tracked and the Uri of the request.

Once the record is successfully inserted, I also register the channel and tracking numbers with the notification hub. The newly inserted record is then picked up by a custom script fired periodically by the WAMS scheduler. The custom script checks for updates to the FOIA request using the Uri column. If there are any updates, it sends out notifications using the notification hub. The benefit of using the notification hub is it scales easily to a large number of recipients without the need for re-architecting the MyFOIA app. Also, the notifications are based on tags. Tags are a way to register MyFOIA app user preferences. In concrete terms, tags in the MyFOIA app are tracking numbers of the FOIA requests that a user is interested in tagging. You can register one or more tags. The notification hub then uses tags to route the notifications.

The final challenge was related to adding social media capabilities to the app. I thought it would be interesting to share "tidbits" from

Figure 7 **Script Function to be Executed from the Currently Loaded HTML**

```
script += "function notifyClick(checkboxControl) {";
script += "var notifyData = '';";
script += "if (checkboxControl.checked) {";
script += "    notifyData = 'add';";
script += "}";
script += "else {";
script += "    notifyData = 'delete';";
script += "}";
script += "window.external.notify(notifyData + '|' + ";
script += " $(checkboxControl).attr('data-number') + '|' + ";
script += " $(checkboxControl).attr('data-uri'));";
script += "}";

script += "var header = $('#curElem thead tr');";
script += "$(header[0]).find('th:last').after('";
script += " <th class=\"detail\">";
script += " <a href=\"#\" onclick=\"return false\">Notify</a>";
script += " </th>');";

script += "var rows = $('#curElem tbody tr');";
script += "for (var i = 0; i < rows.length; i++) {";
script += "    var reqNumber = $(rows[i]).find('td:first a').text();";
script += "    var reqUri = $(rows[i]).find('td:first a').attr('href');";
script +=
  "reqUri = 'https://foiaonline.regulations.gov/foia/action/public/view/' +";
script +=
  "(reqUri.indexOf('request') != -1 ? \"request?\" : \"record?\") + ";
script += "     reqUri.substring(reqUri.indexOf('objectId'));";
script += "    $(rows[i]).find('td:last').after('";
script += "     <td><input type=\"checkbox\" class=\"notifyCheckbox\" ";
script += "     name=\"notify\" value=\"notify\" ";
script += "     data-number=\"' + reqNumber + '\" data-uri=\"' + reqUri + '\" ";
script += "     onclick=\"notifyClick(this);\" /></td>');";
script += "}";

var scriptOperation =
  webviewControl.invokeScriptAsync("eval", new Array(script));
```

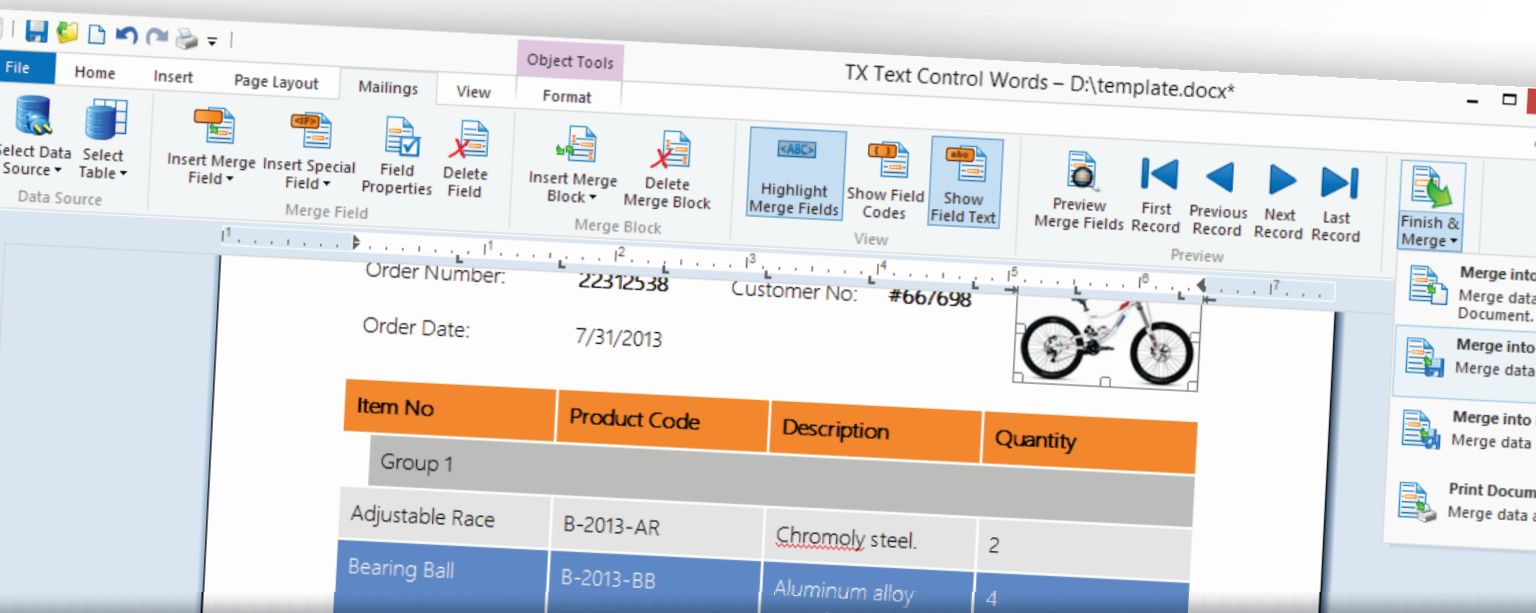
Figure 8 **The onScriptNotify Event Handler**

```
function foiaWebview_onScriptNotify(eventArgs) {
  msgControl.innerText = "Updating Request...";

  var scriptNotifyDataArr = eventArgs.value.split("|");
  var operation = scriptNotifyDataArr[0];
  var reqNumber = scriptNotifyDataArr[1];
  var reqUri = scriptNotifyDataArr[2];

  var requestTable = foiaMobileServiceClient.getTable("Request");

  var tablePromise;

  if (operation == "add") {
    requestTable.insert({
      DeviceId: deviceId,
      TrackingNumber: reqNumber,
      RequestUri: reqUri
  }).done(function () {
    updateRegistration();
  });
  }
  // Code elided for clarity
  }
}

function updateRegistration() {
  var channelOperation =
    pushNotifications.PushNotificationChannelManager.
    createPushNotificationChannelForApplicationAsync();

  channelOperation.then(function (newChannel) {
    channel = newChannel.uri;
    return requestTable.where({ DeviceId: deviceId }).read();
  }).then(function (requests) {
  if (requests.length > 0) {
    var trackingNumArray = getTrackingNumbers(requests);

    return hub.registerApplicationAsync(channel, trackingNumArray);
  }
  // Code elided for clarity
```

TX TEXT CONTROL®
word processing components

Creating a report is as easy as writing a letter

- Reuse MS Word documents as your reporting templates
- Create encrypted and print-ready Adobe PDF and PDF/A
- Royalty-free WYSIWYG template designer
- Powerful and dynamic 2D/3D charting support
- Easy database connections and master-detail nested blocks
- 1D/2D barcode support including QRCode, IntelligentMail, EAN



www.textcontrol.com/reporting

txtextcontrol

US: +1 855-533-TEXT
EU: +49 421 427 067-10

Visual Studio Microsoft Partner

Reporting | Rich Text Editing | Spell Checking | Barcodes | PDF Reflow

FOIA requests and responses over social media tools such as Twitter. Once again, the FOIAonline Web site doesn't currently support such a capability. The challenge in implementing such a capability is that users would be selecting text within the hosted HTML page. How does the app capture the selected text? Fortunately, as shown in **Figure 9**, the WebView control offers a method called captureSelectedContentToDataPackageAsync that captures the selected text and passes it to the loadTwitter method. The loadTwitter method in turn uses the Windows.System.Launcher.launchUriAsync method to launch the default app associated with the URI. It's also interesting to note the use of the Windows.UI.ViewManagement.ViewSizePreference option, introduced in Windows 8.1, to define app view size preference. This lets me launch the browser app alongside the MyFOIA app, as shown in **Figure 4**.

Figure 9 **Capturing Selected Text and Sharing It with Twitter**

```
function mainAppBar_onTweet(eventArgs) {
  msgControl.innerText = "Loading...";
  var captureOperation =
  webviewControl.captureSelectedContentToDataPackageAsync();

  captureOperation.oncomplete = function (completeEvent) {
    var res = completeEvent.target.result;

    if (res) {
      var dataPackage = res.getView();

      dataPackage.getTextAsync().done(function (capturedText) {
        loadTwitter(capturedText);
      });
    }
    else {
      loadTwitter("");
    }
  };
  captureOperation.start();
}
function loadTwitter(tweet) {
  var url = new Windows.Foundation.Uri(
  "https://twitter.com/share?url=https://foiaonline.regulations. "+
  "gov/foia/action/public/home&text=" + tweet);

  var options = new Windows.System.LauncherOptions();
  options.desiredRemainingView =
    Windows.UI.ViewManagement.ViewSizePreference.useMore;

    Windows.System.Launcher.launchUriAsync(url, options).done(
      function (data) {msgControl.innerText = "";
  });
}
```

The FOIA in Action

The following news articles were made possible by use of the FOIA:

- **"A Breach of Truth," *Chattanooga Times Free Press* (Tennessee), March 4, 2006** This article describes how the FOIA allowed the release of a video that shows a briefing conducted for President George W. Bush by Michael Brown, director of the Federal Emergency Management Agency (FEMA). In the video, experts express fears that a hurricane could flood New Orleans with a resulting high death toll, to which President Bush responds "we are fully prepared."
- **"Feds fault Chiron for lax cleanup of flu shot plant," *San Francisco Chronicle*, June 21, 2006** Chiron Corp., a British pharmaceutical company, owned a Liverpool plant that produced 50 percent of the influenza vaccine used by the United States. This article describes how, in 2005, information released under the FOIA led to the discovery that this Chiron plant didn't meet Food and Drug Administration (FDA) regulations. In 2004, the FDA recalled and destroyed the plant's entire production run, leading to a flu vaccine shortage in the winter of that year. The information disclosed showed that Chiron's vaccines were eventually cleared by October 2005, which led to concern for U.S. citizens awaiting the vaccine.
- **"On Range, deadly illness went unreported; Mesothelioma strikes years after victims' exposure to asbestos," *Star Tribune* (Minneapolis, Minn.), Aug. 21, 2005** This article describes how the Mine Safety and Health Administration (MSHA) requested records under the FOIA that exposed a loophole in report requirements that let LTV Steel Mining Co. forego reporting a trend of mesothelioma and other debilitating asbestos-related illnesses among workers in its Minnesota taconite mines dating from 1980. Because mesothelioma usually doesn't appear for more than 20 years after exposure to asbestos, LTV didn't report illnesses and deaths among its retirees, (it *is* required to do so for its active workers). Because of this loophole, there was no action taken to improve safety of other workers at the mine. The failure to report lung disease cases among mine workers was discovered from examination of the documents requested, after reporters spoke with families of affected workers in the Iron Range region. In addition, the MSHA also discovered that the maximum penalty for companies that failed to report an illness was $60.

The exposure of breaches of regulations, suppressed health studies and other information publicized by these articles demonstrates the importance the FOIA plays in keeping citizens informed about the data obtained by federal agencies. By granting the capability for citizens to make requests of any and all information from government agencies, the FOIA provides citizens with the same access to government information.

Stay in the Know

The FOIA is a law that gives citizens the right to access information from the federal government. It's often described as the law that keeps citizens in the know about their government. The MyFOIA Windows Store app is designed to make the FOIA data easily accessible. The MyFOIA app is built using WebView control and WAMS. The WebView control makes it possible to not only take advantage of the existing FOIAonline Web site but also enhance it. WAMS makes it possible for the users to be notified of updates to an FOIA response in which they're interested.

The MyFOIA app is a work in progress, but I've posted the code at github.com/AppliedIS/Foia for you to review.

Finally, I'd like to thank Pamela Steger for help with this article and Sajad Deyargaroo for help with the development of the MyFOIA app. ■

Vishwas Lele *is the CTO at Applied Information Sciences Inc. He is responsible for assisting organizations in envisioning, designing and implementing enterprise solutions. Lele also serves as the Microsoft regional director for the Washington, D.C., area and is a Windows Azure MVP. You can reach him on Twitter at twitter.com/vlele.*

Thanks to the following Microsoft technical experts for reviewing this article: *Kraig Brockschmidt, Kevin Hill and Jake Sabulsky.*

Extreme Performance Linear Scalability

NCache™
100% .NET

For .NET & Java Apps

JvCache™
100% Java

Remove data storage performance bottlenecks and scale your applications to extreme transaction processing (XTP). Cache data in memory and reduce expensive database trips. NCache scales linearly by letting you add inexpensive cache servers at runtime. JvCache is 100% Java implementation of NCache.

Enterprise Distributed Cache

- Extremely fast & linearly scalable with 100% uptime
- Mirrored, Replicated, Partitioned, and Client Cache
- NHibernate & Entity Framework Level-2 Cache

ASP.NET Optimization in Web Farms

- ASP.NET Session State storage
- ASP.NET View State cache
- ASP.NET Output Cache provider
- ASP.NET JavaScript merge/minify

Runtime Data Sharing

- Powerful event notifications for pub/sub data sharing

FREE EXPRESS EDITION

www.alachisoft.com **Download a FREE trial!** **1-925-236-3830**

Rendering PDF Content in Windows Store Apps

Sridhar Poduri

PDF as a document storage and archiving format is well established in today's world. Documents such as books, technical manuals, user guides, reports and more are stored in PDF format. It lets a document be consumed from multiple platforms as long as a supported PDF viewer is available. While viewing PDF documents is largely a nonissue, supporting the rendering of PDF content remains a challenge, especially for Windows Store app developers. With Windows 8.1, Microsoft introduced new APIs that ease the process of rendering PDF content in Windows Store apps.

In this article, I'll look at the different ways to do this rendering. First, I'll focus on the APIs that are part of the Windows Runtime (WinRT) and are accessible to you via JavaScript, C#, Visual Basic .NET and C++. Then I'll focus on the native APIs that let C++ developers render PDF content directly on a DirectX-based drawing surface.

This article discusses:

- Using Windows Runtime APIs to render PDF documents in Windows Store apps
- Opening documents
- Handling password-protected documents
- Customizing rendering
- Using native APIs to render documents on a DirectX surface

Technologies discussed:

Windows Runtime, Windows 8.1, Windows Store Apps

Code download available at:

archive.msdn.microsoft.com/mag201312PDF

The Windows Runtime APIs for PDF Rendering

The Windows Runtime for Windows 8.1 includes a new namespace, Windows.Data.Pdf, which contains the new runtime classes and structures that support PDF rendering in Windows Store apps. In this section, I'll discuss the various classes that make up the Windows.Data.Pdf namespace, used for opening PDF documents, handling password protection, rendering documents, customizing the rendering process and more.

Opening PDF Documents Opening a PDF document programmatically is as easy as calling the static method LoadFromFileAsync from the PdfDocument runtime class. This class is the initial entry point for working with PDF documents. The LoadFromFileAsync method accepts a StorageFile object and begins the process of loading the PdfDocument. Loading a PDF document can sometimes take a long time, so the API returns an asynchronous operation. When the asynchronous operation has completed, you have a valid instance of the PdfDocument object, as shown here:

```
// Obtain a StorageFile object by prompting the user to select a .pdf file
FileOpenPicker openPicker = new FileOpenPicker();
openPicker.ViewMode = PickerViewMode.List;
openPicker.SuggestedStartLocation = PickerLocationId.DocumentsLibrary;
openPicker.FileTypeFilter.Add(".pdf");
StorageFile pdfFile = await openPicker.PickSingleFileAsync();

// Load a PdfDocument from the selected file
create_task(PdfDocument::LoadFromFileAsync(pdfFile)).then(
  [this](PdfDocument^ pdfDoc)
{
  // Handle opened Pdf document here.
});
```

In addition to the LoadFromFileAsync method, the PdfDocument class also contains a helper static method to create a PdfDocument instance from a stream object. If you already hold a reference to a PDF

document as a RandomAccessStream instance, you can simply pass the stream object to the LoadFromStreamAsync method. Depending on your scenario, you can choose to use either the LoadFromFileAsync or LoadFromStreamAsync methods to create a PdfDocument object instance. Once you have a valid PdfDocument instance, you can access individual pages in the document.

With Windows 8.1, Microsoft introduced new APIs that ease the process of rendering PDF content in Windows Store apps.

Handling Password-Protected PDF Documents PDF documents are used to store a wide variety of information, such as credit-card statements or other confidential data. Some publishers don't want users to have unrestricted access to these types of documents and protect them with passwords. Access is granted only to applications whose binaries contain the password. The LoadFromFileAsync and LoadFromStreamAsync methods of the PdfDocument runtime class contain overloaded versions of the methods that accept a password via a String parameter:

```
// Load a PdfDocument that's protected by a password
// Load a PdfDocument from the selected file
create_task(PdfDocument::LoadFromFileAsync(
  pdfFile, "password")).then([this](PdfDocument^ pdfDoc)
{
  Handle opened Pdf document here.
});
```

If you attempt to load a password-protected document without specifying a password, the LoadFromFileAsync and LoadFromStreamAsync methods will throw an exception.

Accessing the Pages in a PDF Document After you create an instance of a PdfDocument object, the Count property will return the number of pages in the PDF document. You can simply iterate from 0 to the "Count – 1" range to get access to individual PDF pages. Each page is of type PdfPage runtime class. The PdfPage runtime class has a method named PreparePageAsync that begins the process of preparing a PDF page for rendering. Page preparation involves parsing and loading the page, initializing Direct2D resources for proper handling of graphic paths and shapes, initializing DirectWrite for handling the correct set of fonts for rendering text and so on. If you don't call PreparePageAsync before beginning to render PDF pages, the render process calls PreparePageAsync for you implicitly. However, you should call PreparePageAsync and have the pages ready for rendering rather than let the rendering process prepare the page. Preparing the page ahead of the actual rendering saves time in the actual rendering process and is a nice optimization technique.

Figure 1 **The PDF App UI**

Figure 2 **The Button_Click Event Handler to Open and Render a PDF Document**

```
void MainPage::Button_Click(Platform::Object^ sender,
  Windows::UI::Xaml::RoutedEventArgs^ args)
{
  m_streamVec->Clear();

  FileOpenPicker^ openPicker = ref new FileOpenPicker();
  openPicker->SuggestedStartLocation = PickerLocationId::DocumentsLibrary;
  openPicker->ViewMode = PickerViewMode::List;
  openPicker->FileTypeFilter->Clear();
  openPicker->FileTypeFilter->Append(L".pdf");

  create_task(openPicker->PickSingleFileAsync())
  .then([this](StorageFile^ pdfFile)
  {
    m_ImagefileName = pdfFile->Name;
    create_task(PdfDocument::LoadFromFileAsync(pdfFile))
    .then([this](PdfDocument^ pdfDoc)
    {
      auto page = pdfDoc->GetPage(0);
      auto stream = ref new InMemoryRandomAccessStream();
      IAsyncAction^ action = page->RenderToStreamAsync(stream);
      auto actionTask = create_task(action);
      actionTask.then([this, stream, page]()
      {
        String^ img_name = m_ImagefileName + ".png";
        task<StorageFolder^> writeFolder(
          KnownFolders::PicturesLibrary->GetFolderAsync("Output"));
          writeFolder
          .then([this, img_name, stream, page](StorageFolder^ outputFolder)
          {
            task<StorageFile^> file(
            outputFolder->CreateFileAsync(img_name,
            CreationCollisionOption::ReplaceExisting));

        file.then([this, stream, page](StorageFile^ file1) {
          task<IRandomAccessStream^> writeStream(
            file1->OpenAsync(FileAccessMode::ReadWrite));
          writeStream.then(
          [this, stream, page](IRandomAccessStream^ fileStream) {
            IAsyncOperationWithProgress<unsigned long long,
            unsigned long long>^ progress
              = RandomAccessStream::CopyAndCloseAsync(
                stream->GetInputStreamAt(0),
                fileStream->GetOutputStreamAt(0));
                auto copyTask = create_task(progress);
                copyTask.then(
                   [this, stream, page, fileStream](
                   unsigned long long bytesWritten) {
                  stream->Seek(0);
                  auto bmp = ref new BitmapImage();
                  bmp->SetSource(fileStream);
                  auto page1 = ref new PdfPageAsImage();
                  page1->PdfPageImage = bmp;
                  m_streamVec->Append(page1);
                  pageView->ItemsSource = ImageCollection;
                  delete stream;
                  delete page;
                  });
                });
            });
          });
        });
    });
  });
}
```

Rendering the PDF Pages Once the PdfPage objects are prepared, they can be rendered. The Rendering API encodes the PdfPage as an image and writes the image data to a stream supplied by the developer. The stream can then either be set as the source for an Image control in the application UI or be used to write the data to disk for use later on.

The rendering happens once you call the RenderToStreamAsync method of the PdfPage runtime class. The RenderToStreamAsync method accepts an instance of an IRandomAccessStream or one of its derived types and writes the encoded data to the stream.

Customizing the Page Rendering The default rendering process involves encoding the PdfPage as a PNG image at the actual dimensions of the PDF page in the document. You can change the default encoding from PNG to either BMP or JPEG, although I strongly recommended that you use PNG encoding for rendering content on-screen because it's lossless and also doesn't generate large bitmaps. However, if you want to generate images from the PDF pages and store them to disk for access later, you should consider using JPEG encoding because it generates smaller image files with an acceptable resolution. You can also specify the SourceRect and DestinationWidth to render only a portion of a PDF page—for example, in response to a zoom-in operation. You can also check for rendering in high-contrast mode by using the Boolean flag IsHighContrastEnabled and setting this flag to true. You can create an instance of the PdfPageRenderOptions structure and pass it to the overloaded version of the RenderToStreamAsync method of the PdfPage class.

The Client Code A simple app demonstrates how easy it is to use these APIs to render PDF content. My sample app (PdfAPISample in the accompanying code download) contains a MainPage with two Button controls, as shown in **Figure 1**.

The click event handlers for both buttons will prompt the user to select a PDF document and render the first page. The event handler for the "Render PDF w/Options" button will use the overloaded RenderToStreamAsync method and change the page background color.

The Button_Click event handler is listed in **Figure 2**.

The Native APIs for PDF Rendering

The WinRT APIs allow easy integration of PDF content in Windows Store apps and are meant to be accessible from all WinRT-supported languages by creating an image file or stream from each page in a PDF document. However, certain classes of Windows Store apps have a requirement to render PDF content directly on-screen. Such apps are usually written using native C++ and utilize XAML or DirectX. For these apps, Windows 8.1 also includes native APIs for rendering PDF content on a DirectX surface.

Figure 3 **Native PDF API App UI**

The native APIs for PDF rendering are present in the windows.data.pdf.interop.h header file and require linking with the windows.data.pdf.lib static library. These APIs are available exclusively for C++ developers who want to render PDF content directly on-screen.

> The default rendering process involves encoding the PdfPage as a PNG image at the actual dimensions of the PDF page in the document.

PdfCreateRenderer The PdfCreateRenderer function is the entry point for the native API. It accepts an instance of an IDXGIDevice as input and returns an instance of an IPdfRendererNative interface. The IDXGIDevice input parameter can be obtained from either a XAML SurfaceImageSource, a VirtualSurfaceImageSource or a XAML SwapChainBackgroundPanel. You might know these are the interop types that XAML supports for mixing DirectX content in a XAML framework. Calling the PdfCreateRenderer function is easy. Make sure to include windows.data.pdf.interop.h and link against the windows.data.pdf.lib static library. Obtain an instance of an IDXGIDevice from the underlying D3DDevice instance. Pass the IDXGIDevice instance to the PdfCreateRenderer function. If the function succeeds, it returns a valid instance of an IPdfRendererNative interface:

```
ComPtr<IDXGIDevice> dxgiDevice;
d3dDevice.As(&dxgiDevice)
ComPtr<IPdfRendererNative> pdfRenderer;
PdfCreateRenderer(dxgiDevice.Get(), &pdfRenderer)
```

The IPdfRendererNative Interface The IPdfRendererNative interface is the only interface supported in the native API. The interface contains two helper methods: RenderPageToSurface and RenderPageToDeviceContext.

RenderPageToSurface is the correct method to use when rendering PDF content to either a XAML SurfaceImageSource or VirtualSurfaceImageSource. The RenderPageToSurface method takes a PdfPage as an input parameter along with an instance of DXGISurface to which to draw the content, an offset on the device to draw and an optional PDF_RENDER_PARAMS structure. As you examine the RenderPageToSurface method, you might be surprised to see the PdfPage input being of type IUnknown. How do you get a PdfPage of type IUnknown? It turns out that with the WinRT API, once you get a valid PdfPage instance from a PdfDocument object, you can use safe_cast to cast a PdfPage to IUnknown.

When you use either the SurfaceImageSource or VirtualSurfaceImageSource interop types, calling BeginDraw returns an offset into the atlas that the XAML framework provides your application to

Figure 4 **The RenderPageRect Method Drawing PDF Content On-Screen**

```
void PageImageSource::RenderPageRect(RECT rect)
{
  m_spRenderTask = m_spRenderTask.then([this, rect]() -> VSISData {
    VSISData vsisData;
    if (!is_task_cancellation_requested())
    {
      HRESULT hr = m_vsisNative->BeginDraw(
        rect,
        &(vsisData.dxgiSurface),
        &(vsisData.offset));
      if (SUCCEEDED(hr))
      {
        vsisData.fContinue = true;
      }
      else
      {
        vsisData.fContinue = false;
      }
    }
    else
    {
      cancel_current_task();
    }
    return vsisData;
  }, m_cts.get_token(), task_continuation_context::use_current())
  .then([this, rect](task<VSISData> beginDrawTask) -> VSISData {
    VSISData vsisData;
    try
    {
      vsisData = beginDrawTask.get();
      if ((m_pdfPage != nullptr) && vsisData.fContinue)
      {
        ComPtr<IPdfRendererNative> pdfRendererNative;
        m_renderer->GetPdfNativeRenderer(&pdfRendererNative);

        Windows::Foundation::Size pageSize = m_pdfPage->Size;
        float scale = min(static_cast<float>(
          m_width) / pageSize.Width,
          static_cast<float>(m_height) / pageSize.Height);

          IUnknown* pdfPageUnknown = (IUnknown*)
          reinterpret_cast<IUnknown*>(m_pdfPage);

          auto params = PdfRenderParams(D2D1::RectF((rect.left / scale),
            (rect.top / scale),
            (rect.right / scale),
            (rect.bottom / scale)),
            rect.right - rect.left,
            rect.bottom - rect.top,
            D2D1::ColorF(D2D1::ColorF::White),
            FALSE
            );
          pdfRendererNative->RenderPageToSurface(
            pdfPageUnknown,
            vsisData.dxgiSurface.Get(),
            vsisData.offset, &params);
      }
    }
    catch (task_canceled&)
    {

    }
    return vsisData;
  }, cancellation_token::none(), task_continuation_context::use_arbitrary())
  .then([this](task<VSISData> drawTask) {
    VSISData vsisData;
    try
    {
      vsisData = drawTask.get();
      if (vsisData.fContinue)
        m_vsisNative->EndDraw();
    }
    catch (task_canceled&)
    {

    }
  }, cancellation_token::none(), task_continuation_context::use_current());
}
}
```

draw content. You should pass that offset to RenderPageToSurface to ensure that drawing happens at the correct position.

The RenderPageToDeviceContext is the correct method to use when rendering PDF content to a XAML SwapChainBackgroundPanel. The RenderPageToDeviceContext takes a PdfPage as an input parameter along with an instance of ID2D1DeviceContext to which to draw the content and an optional PDF_RENDER_PARAMS structure.

In addition to drawing directly on-screen, the rendering can also be customized by using the PDF_RENDER_PARAMS structure. Both the RenderPageToSurface and RenderPageToDeviceContext accept an instance of PDF_RENDER_PARAMS. The options provided in PDF_RENDER_PARAMS are similar to the WinRT PDFPageRenderOptions structure. Note that neither of the native APIs are asynchronous methods. The rendering happens directly on-screen without incurring the cost of encoding and decoding.

While the WinRT APIs encode the PDF page as an image and write the image file to disk, the native APIs use a DirectX-based renderer to draw the PDF content on-screen.

Again, a simple app demonstrates how to use these APIs to render PDF content. This sample app (DxPdfApp in the accompanying code download) contains a MainPage with one button control, as shown in **Figure 3**.

The code to open a document and access a PDF page remains the same between the WinRT API and the native API. The major change is in the rendering process. While the WinRT APIs encode the PDF page as an image and write the image file to disk, the native APIs use a DirectX-based renderer to draw the PDF content on-screen, as shown in **Figure 4**.

Learn More

I discussed the WinRT PDF API in Windows 8.1 that lets you incorporate PDF content in Windows Store apps. I also discussed the differences between using the WinRT API or the C++/DirectX API, and the benefits of each approach. Finally, I provided a set of recommendations on which API should be used under certain scenarios. For more information on the PDF APIs in Windows 8.1, check out the documentation and PDF viewer showcase sample on MSDN at bit.ly/1bD72T0. ■

Sridhar Poduri *is a program manager at Microsoft. A C++ aficionado and author of the book, "Modern C++ and Windows Store Apps" (Sridhar Poduri, 2013), he blogs regularly about C++ and the Windows Runtime at sridharpoduri.com.*

Thanks to the following technical expert for reviewing this article: *Subramanian Iyer (Microsoft)*

TED NEWARD

Getting Started with Oak: A Different Approach

As I discussed last time, the Oak project is a Web framework that incorporates dynamic aspects and approaches common to more dynamic-language-based frameworks (such as Ruby on Rails or any of the various MVC Web frameworks in Node.js, such as Express or Tower). Because it's based on the Microsoft .NET Framework and uses dynamic and Dynamic Language Runtime (DLR) parts of C#, Oak takes quite a different approach to developing Web applications from those used by the traditional ASP.NET MVC developer. For that reason, as you discovered last time, getting the necessary bits to do Oak development is a little more sophisticated than just pulling it down via NuGet.

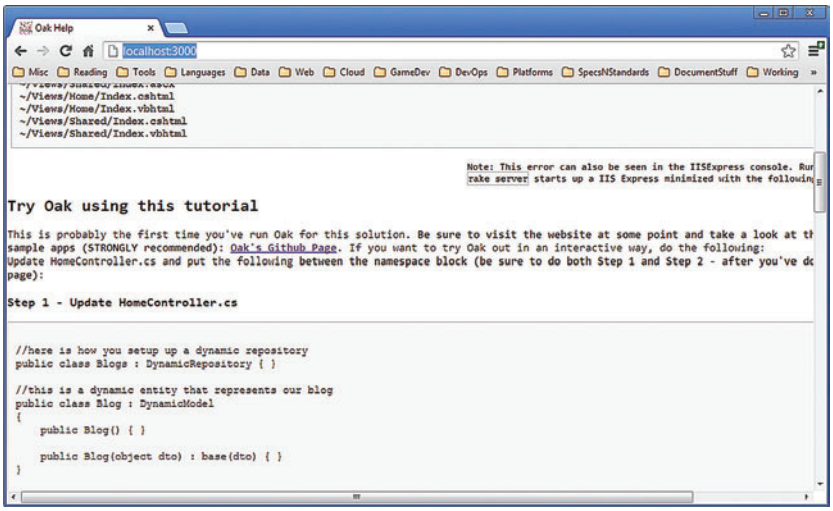


Figure 1 **Oak Project Help Window**

Assuming that you've read the previous column (msdn.microsoft.com/magazine/dn451446), downloaded the bits, installed them, kicked off the continuous build sidekick and gotten the initial build running on your machine (in IIS Express, on port 3000, as you recall), it's time to start working on developing in Oak.

Firing Up

If it isn't running on your box, do a "rake" and "rake server" from a command prompt, just to make sure that everything is kosher. Then launch "sidekick" (if it isn't already running), and open a browser to localhost:3000, as shown in **Figure 1**.

As the resulting tutorial implies, there's a sort of breadcrumbs-style walkthrough to learn Oak. Before I get into it, though, take a quick look at the project structure, shown in **Figure 2**.

The seed project consists of two projects: the ASP.NET MVC project and the project with the tests for the solution. The MVC project is a traditional ASP.NET MVC app, with the added "Oak" folder containing the source files making up the Oak parts of the project. This makes it absolutely trivial to step through the Oak parts of the code during debugging, and, in the spirit of all open source projects, also enables local modifications if and when necessary. Currently, the project has no models, three controllers and just a couple of views. More important, because there's not much by way of code here, the first request to the endpoint yields an error that no "Index.cshtml" (or similar) view exists. The Oak bootstrapper suggests two steps. First, you need to create two new types: Blog and Blogs, a collection of Blog instances. Calling the class "Blogs" gives you easy, convention-based access to the Blogs table in the database:

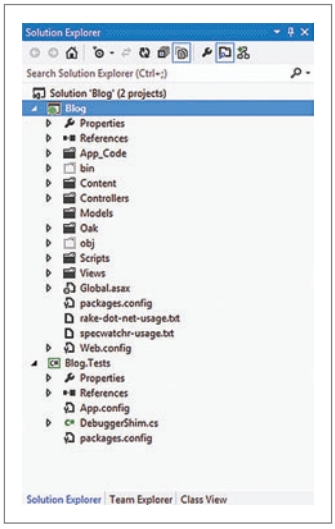


Figure 2 **Oak Project Structure in Visual Studio Solution Explorer**

```
public class Blogs : DynamicRepository
{
}

// This is a dynamic entity that represents the blog
public class Blog : DynamicModel
{
  public Blog() { }

  public Blog(object dto) : base(dto) { }
}
```

Second, the HomeController needs some changes to be able to respond to different HTTP requests sent, as shown in **Figure 3**.

Much of this will be familiar to ASP.NET developers. What's strikingly different is that everything is typed as the C# dynamic type, not as Blog or Blogs instances. The Blog type itself has no fields or properties. The Blogs type—an aggregation of Blog instances—similarly has no code declared on it to insert, remove, list, replace or do any of the other operations commonly associated with collections.

SpreadsheetGear

Performance Spreadsheet Components

SpreadsheetGear 2012 Now Available

NEW!

WPF and Silverlight controls, multithreaded recalc, 64 new Excel compatible functions, save to XPS, improved efficiency and performance, Windows 8 support, Windows Server 2012 support, Visual Studio 2012 support and more.

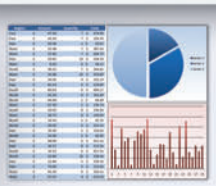
Excel Reporting for ASP.NET, WinForms, WPF and Silverlight

Easily create richly formatted Excel reports without Excel from any ASP.NET, Windows Forms, WPF or Silverlight application using spreadsheet technology built from the ground up for performance, scalability and reliability.

Excel Compatible Windows Forms, WPF and Silverlight Controls

Add powerful Excel compatible viewing, editing, formatting, calculating, filtering, charting, printing and more to your Windows Forms, WPF and Silverlight applications with the easy to use WorkbookView controls.

Excel Dashboards, Calculations, Charting and More



You and your users can design dashboards, reports, charts, and models in Excel or the SpreadsheetGear Workbook Designer rather than hard to learn developer tools and you can easily deploy them with one line of code.

Free 30 Day Trial

Download our fully functional 30-Day evaluation and bring Excel Reporting, Excel compatible charting, Excel compatible calculations and much more to your ASP.NET, Windows Forms, WPF, Silverlight and other Microsoft .NET Framework solutions.

www.SpreadsheetGear.com

Visual Studio Microsoft Partner

SpreadsheetGear

Toll Free USA (888) 774-3273 | Phone (913) 390-4797 | sales@spreadsheetgear.com

Figure 3 **Responding to Different HTTP Requests**

```
public class HomeController : Controller
{
  // Initialize the blog
  Blogs blogs = new Blogs();

  public ActionResult Index()
  {
    // Return all blogs from the database
    ViewBag.Blogs = blogs.All();
    return View();
  }

  // Controller action to save a blog
  [HttpPost]
  public ActionResult Index(dynamic @params)
  {
    dynamic blog = new Blog(@params);
```

Figure 4 **Creating the View**

```
@{
  ViewBag.Title = "Index";
}

<h2>Hello World</h2>
<div>
If this page came up successfully, you're doing well! Go ahead and create
a blog (try to create blogs with duplicate names).
</div>
<br />

@using (Html.BeginForm())
{
  @Html.TextBox("Name")
  <input type="submit" value="create" />
}

@foreach (var blog in ViewBag.Blogs)
{
  <div>
    <pre>@blog</pre>
    <br />
    <div>
    Almost there, you have comments listing; let's try to add one.
    </div>
    <br />
  </div>
}
```

A lot of this power comes from the dynamic-based Gemini library, a core part of the Oak project (and the subject of my August 2013 column, "Going Dynamic with the Gemini Library," at msdn.microsoft.com/magazine/dn342877). Blog extends the DynamicModel base class, which essentially means you can program against it without having to define the model up front. Any field you reference on any given Blog instance will be there, even if you've never referenced it before. This is the power of dynamic programming. Similarly, Blogs is a DynamicRepository. As such, it already knows how to store and manipulate DynamicModel objects.

> Even more interesting is that right out of the box, Oak knows how to store Blog instances to a SQL table.

Even more interesting is that right out of the box, Oak knows how to store Blog instances to a SQL table (named, not surprisingly, "Blogs"). Not only will it create the database schema on first use, but it can "seed" the database with any startup data the system might need.

When you construct a Blog instance, it takes a dynamic object as a parameter. The base class constructor knows how to walk through any of the dynamically defined fields/properties on that object. Similarly, the Blogs instance will also know how to iterate through all dynamically defined fields/properties of a Blog instance. It will store them to the database (on the call to blog.Insert). It also knows how to retrieve a Blog instance from the database via the BlogId field. All of this is powered by the code in **Figure 3**; no additional model code is necessary—at least, not yet. (There are other things you'll want there, but for now, it all just works.)

By the way, if you're wondering what the @ operator is, remember params is actually a reserved word in C#. In order to use it as an acceptable parameter name, you have to prefix it with @ to tell the C# compiler to not treat it as a keyword.

Having modified HomeController.cs, the next step is to create an acceptable view, Index.cshtml, in a Home folder under the Views folder. This will display the results of the controller's work, as shown in **Figure 4**.

On the surface, the view isn't just another ASP.NET view. Again, the dynamic nature of the system comes into play. No Blog instance has defined a Name field on the Blog type, yet when the form at

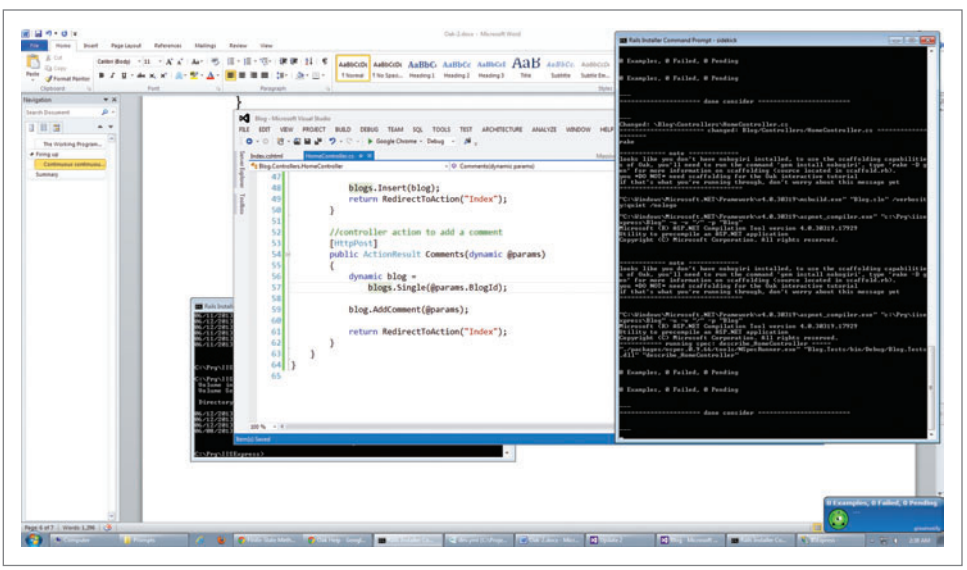


Figure 5 **A Growl Notification Showing the Build Succeeded**

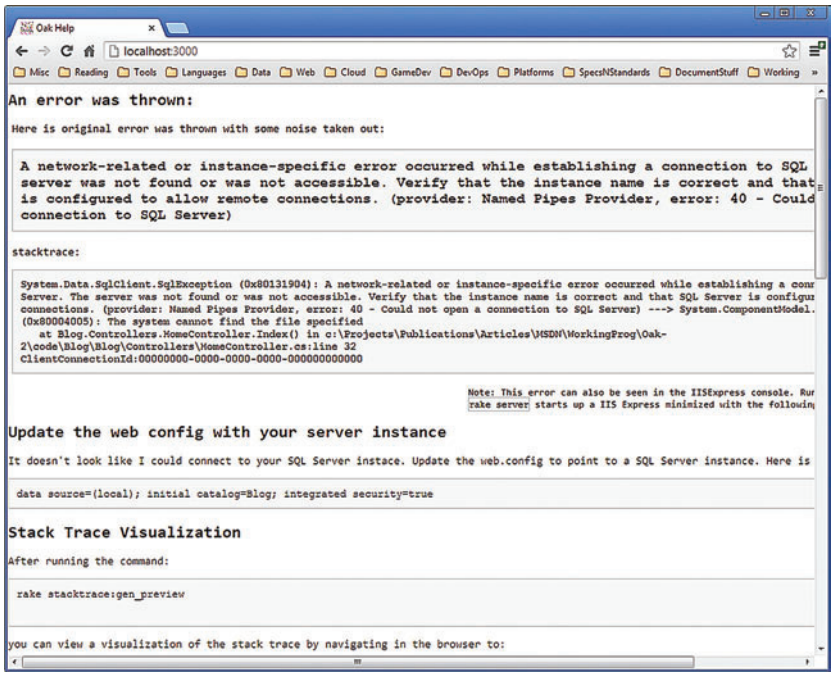


Figure 6 **Oak Cannot Make Bricks Without Clay**

the top of the view is submitted, a "name=..." parameter will be passed in to the HomeController. This controller will then pass on that name/value pair in the @params variable used to initialize a Blog instance. Without any additional work on your part, the Blog instance now has a Name field/property on it.

Continuous Continuousness

By the way, if you're playing the home version of this game and you saved these files, you'll see something interesting happened (see **Figure 5**).

First of all, a notification popped up in the lower-right corner. This is Growl at work. It's giving you the green-light notification that the build kicked off by the sidekick app you launched earlier has succeeded. If it fails for some reason, the graphic on the left of the notification window will be red, and the console output will display in the notification. Second, if you look in the console window in which sidekick is running, it will be apparent what happened. The filesystem watcher in the Blog directory registered that a source file changed (because you saved it). The sidekick took that as a cue to rebuild the project.

Assuming the files are saved and the code was correct, hitting localhost:3000 again yields the new result, shown in **Figure 6**.

This time, Oak is trying to connect to a running SQL Server instance in order to fetch any data from that table (the Blog table). This is Oak automatically trying to manage the object-relation mapping (ORM) parts of the project on your behalf, and I'll get further into that next time.

A Different Style

As you can see, using Oak definitely involves a different style of developing. At no point did you have to do anything more complicated in Visual Studio than open a file, change its contents and save the new version—and add a new file to the project. At no point did you ever kick off a build, run it from within Visual Studio, or open Server Explorer to create tables or kick off SQL Server scripts.

All those things are still available to you, should the need arise. Again, Oak is just a layer (and a fairly thin one at that) on top of the traditional ASP.NET stack. However, this layer permits a degree of rapid development and productivity similar to other more dynamic environments, without losing the things you like from the statically typed world.

There's more to do with Oak, but that will remain a subject for future discussion.

Happy coding! ■

Ted Neward *is the principal of Neward & Associates LLC. He has written more than 100 articles and authored and coauthored a dozen books, including "Professional F# 2.0" (Wrox, 2010). He's an F# MVP and speaks at conferences around the world. He consults and mentors regularly—reach him at ted@tedneward.com if you're interested in having him come work with your team, or read his blog at blogs.tedneward.com.*

Thanks to the following technical expert for reviewing this article: *Amir Rajan (Oak project creator)*

RACHEL APPEL

Everything You Need to Know About the WinJS ListView Control

You have data. Lots of data. You need to present this data in such a way that users can effortlessly access and make sense of it while in your app. Apps expose their data in the form of news articles, recipes, sports scores, financial charts and more, all parading in various-sized compartments across the screen and trying to attract the attention of the consumer. The vast majority of apps on the market today present data in a grid or list format because it makes sense, as small to mid-size grids with data are easy for humans to consume, search and filter. From enterprise apps to personal apps to whatever the app may be, grids are the scaffolding that props up data for quick visual skimming.

In Windows Store apps, you can set up this structure for data presentation by using the ListView control. If you're new to Windows Store app development, you can get up to speed by reading my February 2013 article, "Create Windows Store Apps with HTML5 and JavaScript" (msdn.microsoft.com/magazine/jj891058) and my July 2013 article, "Mastering Controls and Settings in Windows Store Apps Built with JavaScript" (msdn.microsoft.com/magazine/dn296546).

ListView Control Basics

Available in both HTML and XAML, the ListView is the control du jour for presenting data in a grid or list format. In Windows Library for JavaScript (WinJS) apps (the focus of this article), you can use the ListView control by setting the data-win-control attribute on a host element to "WinJS.UI.ListView," like so:

```
<div id="listView" data-win-control= "WinJS.UI.ListView"></div>
```

The that hosts the ListView contains no child elements. However, it does contain basic configuration information in an attribute named data-win-options. Data-win-options lets you set any property of the ListView control using a declarative syntax in the HTML page. To use the ListView properly, you'll need to apply the following characteristics to it:

- The group and item templates for the ListView.
- The group and item data sources of the ListView.
- Whether the ListView uses a grid or list layout (the default is grid).

You should also specify whether the ListView's item selection mode is single or multiple (the default is multiple). A basic ListView with the layout and selectionMode properties set in the data-win-options attribute looks like this:

```
<div id="listView" data-win-control= "WinJS.UI.ListView" data-win-options=
  "{ selectionMode: 'single', layout : {type: WinJS.UI.GridLayout} }" ></div>
```

This article discusses a prerelease version of Windows 8.1. All related information is subject to change.

Figure 1 **The Header and Item Templates for the ListView Control**

```
<div class="headertemplate" data-win-control="WinJS.Binding.Template">
  <button class="group-header win-type-x-large win-type-interactive"
    data-win-bind="groupKey: key" onclick="Application.navigator.pageControl
    .navigateToGroup(event.srcElement.groupKey)" role="link" tabindex="-1"
    type="button">
    <span class="group-title win-type-ellipsis" data-win-bind=
      "textContent: title"></span>
    <span class="group-chevron"></span>
  </button>
</div>
<div class="itemtemplate" data-win-control="WinJS.Binding.Template">
  <div class="item">
    <img class="item-image" src="#" data-win-bind=
      "src: backgroundImage; alt: title" />
      <div class="item-overlay">
        <h4 class="item-title" data-win-bind="textContent: title"></h4>
        <h6 class="item-subtitle win-type-ellipsis" data-win-bind=
          "textContent: subtitle"></h6>
      </div>
  </div>
</div>
```

Though the preceding code defines a ListView, the ListView doesn't work all by itself. It needs the help of the WinJS.Binding.List object. The List object binds arrays filled with objects to the HTML elements defined in the item and group templates. This means that the List object defines the data to display while the template defines how to display it.

Create ListView Templates

Once you have the for the ListView set up, you can move on to creating the templates for it. The ListView depends on HTML templates to display data that's readable to the user. Luckily, the Grid, Split and Hub (the Hub is available in Windows 8.1) Windows Store app templates contain everything you need to present data in a grid or list format, including sample data, predefined ListView controls and predefined CSS classes. You can modify these templates or go ahead and create your own if you'd like. Note, however, that if you create your own templates, you should adhere to the principles of modern UI design and implement the Windows 8 silhouette as described in the Dev Center for Windows Store apps at bit.ly/IkosnL. This is done for you when you use the built-in Visual Studio templates.

The ListView requires an item template, and if you're grouping data, then it needs a header template as well. The parent elements of the item and group templates are simple elements with the data-win-control attribute set to "WinJS.Binding.Template."

The header template should contain links for each group that, when clicked, take the user to a page that lists items belonging to that

group. This is an example of a rather common master/detail navigation pattern. In **Figure 1**, the classed as "headertemplate" contains a <button> element bound to the group's key. When the user taps or clicks the button, she moves to a page revealing the members of that group.

The item template in **Figure 1** consists of tags that enclose an image and two text fields. Much of today's data found in modern apps is graphic, so there's an <img> element inside the item template. The sample data fills this element with an image that's just a solid gray color. **Figure 2** depicts the default ListView from the Grid Layout.

After coding the item and group templates, it's time to hook them up to some data.

Data and Data Binding with the ListView Control

JavaScript and JSON go hand in hand (JSON is JavaScript Object Notation, after all), so once you've retrieved some JSON data, just stuff it into an array and the Windows.Binding.List object turns it into a usable data source for the ListView. In other words, you don't directly tie the ListView to an array or data source, meaning the List object serves as an intermediary between the ListView and the data source. This is because the List object transforms the data into something the ListView knows how to use—the ListView itself only defines the look and layout of the grid. The List object also provides methods for searching, sorting, adding and deleting members of the underlying array.

Examining the \js\data.js file uncovers a Data namespace as well as the arrays making up the sample data. The crucial takeaway is that the two arrays blend to form a master/detail relationship. The group property of each object in the sampleItems array (details) refers to its group in the sampleGroups array (master), as shown in **Figure 3**.

Of course, you'll replace the sample data with your own by building your own arrays, accessing JSON or XML data, or perhaps by calling a Web service. You aren't tied to using the Data namespace and instead can define your own..

Near the top of the data.js file is the following line of code:

```
var list = new WinJS.Binding.List();
```

This list variable is a container for an array. You can add an array to the List by passing it into the List's constructor method or by using the push method:

```
generateSampleData().forEach(function (item) {
  list.push(item);
});
```

Doing this fills the list with the array data, and you're now ready to associate the list and the ListView. If you're sticking with the default template code, you should perform this association when the app first loads, in the _initializeLayout function of the \js\default.js file, as shown here:

```
listView.itemDataSource = Data.items.dataSource;
listView.groupDataSource = Data.groups.dataSource;
```

Of course, depending on the size of your data, the load time may vary, so you might need to modify the loading process. Use your best judgment about loading data into memory, keeping in mind the importance of performance to users.

Notice that the item and group data sources are set to the Data.items.dataSource and Data.groups.dataSource, respectively. The members named "items" and "groups" of the Data namespace refer back to the functions that have created the array containing the data (that is, groupedItems). The Data namespace declaration in the \js\data.js file reflects this notion and shows other public members in the namespace for working with data:

```
WinJS.Namespace.define("Data", {
  items: groupedItems,
  groups: groupedItems.groups,
  getItemReference: getItemReference,
  getItemsFromGroup: getItemsFromGroup,
  resolveGroupReference: resolveGroupReference,
  resolveItemReference: resolveItemReference
});
```

The items and groups members of the Data namespace point to the groupedItems object, which has constructed the data properly. Everything in the Data namespace you've seen so far is included in the Visual Studio project templates. If you choose to start with a blank project, you'll need to mold the data yourself by creating similar data-access methods instead of relying on the Data namespace members.

At this point, the ListView is complete with data and bindings set up. You can bind properties of the objects in the data source to HTML elements by using the data-win-bind attribute, as shown here:

```
<h4 class="item-title" data-win-bind="textContent: title"></h4>
```

The preceding line of code binds the title property to the <h4> element as part of its text. **Figure 1** and **Figure 2** have more samples of the data-win-bind attribute in action.

Now that you have the ListView and data access ready, it's time to move on to styling the ListView.

Style the ListView Control

Presentation is all about style. The WinJS libraries contain a complete set of CSS rules with predefined styles you can overwrite to mold the ListView in a variety of ways. If you're unfamiliar with styling WinJS controls, see my October 2013 article, "Build a Responsive and Modern UI with CSS for WinJS Apps," at msdn.microsoft.com/magazine/dn451447. You can style the entire ListView by overwriting the .win-listview CSS class. Along with styling the ListView, you can set the constituent pieces of the ListView using the following class selectors:

- .win-viewport: Styles the ListView's viewport. The viewport is where the scrollbar sits.

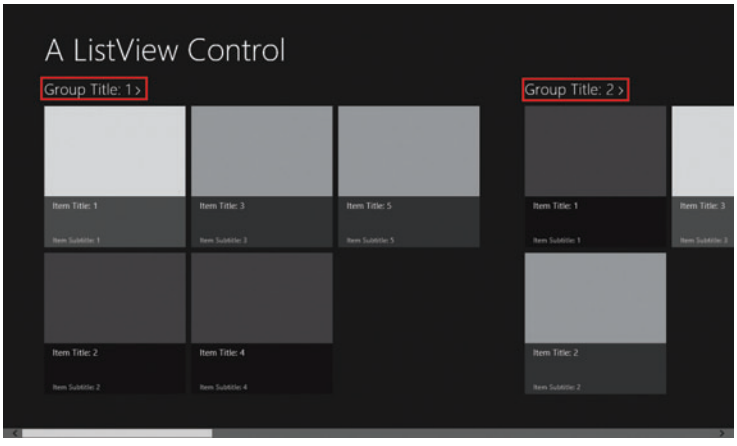


Figure 2 **The Default ListView from the Grid Template, with Heading and Navigation Buttons Marked in Red**

- .win-surface: Styles the scrollable area of the ListView. This area moves when a user scrolls.

There are two ways to style items in a ListView. You can apply styles to the item template via the .win-item class, or you can override the .win-container class. Keep in mind that each item in a ListView comprises multiple HTML elements (refer to **Figure 1** to view these elements). As you can see from **Figure 1**, the elements that make up the item template contain an .item, .item-image, .item-overlay, .item-title and .item-subtitle class. You'll find none of these defined in the system style sheets (that is, ui-light and ui-dark), as these are for you to style.

You should be aware of a handful of gotchas involved with styling, especially concerning when to apply margins and padding to the ListView control. You can review all the ins and outs of styling the ListView in the Dev Center for Windows Store apps at bit.ly/HopfUg. Don't forget to create styles for all the various view states that your app might need.

Windows 8.1 includes some styling changes to the ListView concerning child/descendant selector specificity. This is because a new node belongs to the internal tree structure of the page, so you must update your CSS selectors to contain the .win-itembox class selector, like this: .win-container | .win-itembox | .win-item.

Respond to View State Changes in the ListView

Responding to the new Windows 8 snap and filled views is important for passing the Windows Store certification process. The snap view, along with the full and filled views, is how users can arrange multiple Windows Store apps on the screen. In Windows 8, users may resize up to two open app windows, one in filled view and one in snap view. In Windows 8.1, the maximum number of windows increases to four and there are more options for app display. These views are called tall or narrow in Windows 8.1, and are slightly different than the snap and filled views of Windows 8.

This means you must code the ListView control to change its format in response to changes in app view states. This is a process called responsive design, and you can achieve it by using CSS media queries. For a primer on CSS media queries, see my blog post, "Create mobile site layouts with CSS Media Queries," at bit.ly/1c39mDx.

Figure 3 **Sample Data in Array Form in the Grid Project Template**

```
var sampleGroups = [
  { key: "group1", title: "Group Title: 1", subtitle: "Group Subtitle: 1",
    backgroundImage: darkGray, description: groupDescription },
  { key: "group2", title: "Group Title: 2", subtitle: "Group Subtitle: 2",
    backgroundImage: lightGray, description: groupDescription },
  { key: "group3", title: "Group Title: 3", subtitle: "Group Subtitle: 3",
    backgroundImage: mediumGray, description: groupDescription }
];
var sampleItems = [
  { group: sampleGroups[0], title: "Item Title: 1",
    subtitle: "Item Subtitle: 1", description: itemDescription,
    content: itemContent, backgroundImage: lightGray },
  { group: sampleGroups[0], title: "Item Title: 2",
    subtitle: "Item Subtitle: 2", description: itemDescription,
    content: itemContent, backgroundImage: darkGray },
  { group: sampleGroups[0], title: "Item Title: 3", subtitle:
    "Item Subtitle: 3", description: itemDescription,
    content: itemContent, backgroundImage: mediumGray },
  { group: sampleGroups[1], title: "Item Title: 1", subtitle:
    "Item Subtitle: 1", description: itemDescription,
    content: itemContent, backgroundImage: darkGray },
  { group: sampleGroups[2], title: "Item Title: 2", subtitle:
    "Item Subtitle: 2", description: itemDescription,
    content: itemContent, backgroundImage: mediumGray },
];
```

Media queries shape the ListView control to fit different view states in a way that makes sense for the varying screen sizes and orientations that go with each view. When switching to tall views, the ListView needs to turn into a list, as shown here:

```
listView.layout = new ui.ListLayout();
```

Later, when the user switches back to the original view state, you must set the ListView back to a grid:

```
listView.layout = new ui.GridLayout({ groupHeaderPosition: "top" });
```

If you want to change styling in the items in the ListView when the screen changes, add CSS to this media query in the \css\default.css file:

```
@media screen and (-ms-view-state: snapped) {...}
```

You don't need a media query for full or filled views, as those use the default style sheets. However, you can use different media queries for a variety of screen sizes if needed.

ListView and Semantic Zoom

The reimagining of Windows in version 8 entails new ways to visualize, navigate and search data. This means you need to think differently about how to approach search. Instead of users having to type phrases into search boxes and sift through lists of results, they can now use semantic zoom to condense the data into digestible sections.

Semantic zoom lets the user search for things by using a pinch gesture (or Ctrl + mouse wheel) to pan or zoom out and observe the data in an aggregate format. For example, the Windows Start page behaves this way by showing users all available apps when they zoom out. Using semantic zoom in your app is easy, because it's just a control for WinJS:

```
<div data-win-control="WinJS.UI.SemanticZoom">
  <!-- The control that provides the zoomed-in view goes here. -->
  <!-- The control that provides the zoomed-out view goes here. -->
</div>
```

The SemanticZoom control is simply a wrapper for a ListView or two, or perhaps the HTML Repeater control new to Windows 8.1.

Odds and Ends About the ListView

Don't use the ListView as a general-purpose layout control. Use the CSS box model for that instead. In Windows 8.1, you should consider whether you're better off using a ListView control or a Repeater control. A Repeater control is better if you don't require a lot of functionality from the control and just need to repeat the same HTML design multiple times. At the time of this writing, Windows 8.1 is in preview, so there could be a few other changes to the ListView as well as other Windows Store app API components. For more information on Windows 8.1 API changes, see the Dev Center documentation at bit.ly/1dYTylx. ■

Rachel Appel *is a consultant, author, mentor and former Microsoft employee with more than 20 years of experience in the IT industry. She speaks at top industry conferences such as Visual Studio Live!, DevConnections, MIX and more. Her expertise lies within developing solutions that align business and technology focusing on the Microsoft dev stack and open Web. For more about Appel, visit her Web site at rachelappel.com.*

Thanks to the following technical expert for reviewing this article: *Eric Schmidt (Microsoft)*

Work With Relational Data, Not Complex APIs or Services

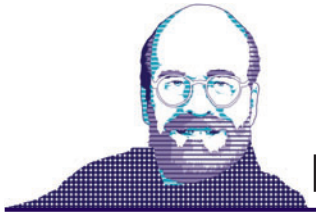
Whether you are a developer using ADO.NET, JDBC, OData, or MySQL, or a systems integrator working with SQL Server or Biztalk, or even an information worker familiar with ODBC or Excel – our products give you bi-directional access to live data through easy-to-use technologies that you are already familiar with. If you can connect to a database, then you will already know how to connect to Salesforce, SAP, SharePoint, Dynamics CRM, Google Apps, QuickBooks, and much more!

Give RSSBus a try today and see what mean:

visit us online at www.rssbus.com to learn more or download a free trial.

rssbus

INTEGRATION YOUR WAY



CHARLES PETZOLD

Character Outline Geometries Gone Wild

The most significant advance in digital typography on personal computers occurred more than 20 years ago with the switch from bitmap fonts to outline fonts. In versions of Windows prior to Windows 3.1, onscreen text was generated from font files consisting of tiny bitmaps of specific point sizes. These bitmaps could be scaled for in-between sizes, but not without a loss of fidelity.

Adobe Systems Inc. pioneered an alternative approach to displaying computer fonts with PostScript, which defined font characters with graphic outlines consisting of straight lines and Bézier curves. You could scale these outlines to any dimension, and algorithmic "hints" helped preserve fidelity at small point sizes. As an alternative to PostScript for fonts on the personal computer screen, Apple Inc. developed the TrueType font specification, which Microsoft later adopted. That eventually evolved into today's common OpenType standard.

> These days, we take for granted the continuous scalability of onscreen fonts, as well as the ability to rotate or skew text using graphical transforms.

These days, we take for granted the continuous scalability of onscreen fonts, as well as the ability to rotate or skew text using graphical transforms. Yet it's also possible to obtain the actual geometries that define the outlines of these font characters and use them for unusual purposes, such as outlining text characters, or clipping, or performing non-linear transforms.

From Font to Clipping Geometry

If you want to obtain character outline geometries in a Windows Store application, the Windows Runtime API won't help. You'll have to use DirectX. The GetGlyphRunOutline method of IDWriteFontFace writes the character outlines into an IDWriteGeometrySink (which is the same as an ID2D1SimplifiedGeometrySink) that defines (or contributes to) an ID2D1PathGeometry object.

Code download available at archive.msdn.microsoft.com/mag201312DXF.

Figure 1 shows the constructor of a rendering class in a Windows 8.1 application named ClipToText that I created from the DirectX App (XAML) template. The project includes the distributable Miramonte Bold font file, and the code shows how to convert a glyph run to a path geometry. As usual, I've removed the checks of errant HRESULT values for purposes of clarity.

Although the code in **Figure 1** obtains an IDWriteFontFace object from a privately loaded font, applications can also obtain font face objects from fonts in font collections, including the system font collection. The code in **Figure 1** specifies glyph indices explicitly corresponding to the text "CLIP," but you can also derive glyph indices from a Unicode character string using the GetGlyphIndices method.

Figure 1 **Converting a Glyph Run to a Path Geometry**

```
ClipToTextRenderer::ClipToTextRenderer(
  const std::shared_ptr<DeviceResources>& deviceResources) :
  m_deviceResources(deviceResources)
{
  // Get font file
  ComPtr<IDWriteFactory> factory = m_deviceResources->GetDWriteFactory();
  String^ filePath = Package::Current->InstalledLocation->Path +
    "\\Fonts\\Miramob.ttf";
  ComPtr<IDWriteFontFile> fontFile;
  factory->CreateFontFileReference(filePath->Data(), nullptr, &fontFile);

  // Get font face
  ComPtr<IDWriteFontFace> fontFace;
  factory->CreateFontFace(DWRITE_FONT_FACE_TYPE_TRUETYPE,
                          1,
                          fontFile.GetAddressOf(),
                          0,
                          DWRITE_FONT_SIMULATIONS_NONE,
                          &fontFace);

  // Create path geometry and open it
  m_deviceResources->GetD2DFactory()->CreatePathGeometry(&m_clipGeometry);

  ComPtr<ID2D1GeometrySink> geometrySink;
  m_clipGeometry->Open(&geometrySink);

  // Get glyph run outline ("CLIP")
  uint16 glyphIndices [] = { 0x0026, 0x002F, 0x002C, 0x0033 };
  float emSize = 96.0f;      // 72 points, arbitrary in this program

  fontFace->GetGlyphRunOutline(emSize,
                               glyphIndices,
                               nullptr,
                               nullptr,
                               ARRAYSIZE(glyphIndices),
                               false,
                               false,
                               geometrySink.Get());

  // Don't forget to close the geometry sink!
  geometrySink->Close();

  CreateDeviceDependentResources();
}
```

Figure 2 **Clipping with a Path Geometry**

```
bool ClipToTextRenderer::Render()
{
  if (!m_needsRedraw)
    return false;

  ID2D1DeviceContext* context = m_deviceResources->GetD2DDeviceContext();
  Windows::Foundation::Size outputBounds = m_deviceResources->GetOutputBounds();

  context->SaveDrawingState(m_stateBlock.Get());
  context->BeginDraw();
  context->Clear(ColorF(ColorF::DarkBlue));

  // Get the clip geometry bounds
  D2D_RECT_F geometryBounds;
  m_clipGeometry->GetBounds(D2D1::IdentityMatrix(), &geometryBounds);

  // Define transforms to center and scale clipping geometry
  Matrix3x2F orientationTransform =
    m_deviceResources->GetOrientationTransform2D();
  Matrix3x2F translateTransform =
    Matrix3x2F::Translation(SizeF(-geometryBounds.left, -geometryBounds.top));

  float scaleHorz = outputBounds.Width /
                    (geometryBounds.right - geometryBounds.left);
  float scaleVert = outputBounds.Height /
                    (geometryBounds.bottom - geometryBounds.top);
  Matrix3x2F scaleTransform = Matrix3x2F::Scale(SizeF(scaleHorz, scaleVert));

  // Set the geometry for clipping
  ComPtr<ID2D1Layer> layer;
  context->CreateLayer(&layer);

  context->PushLayer(
    LayerParameters(InfiniteRect(),
                    m_clipGeometry.Get(),
                    D2D1_ANTIALIAS_MODE_PER_PRIMITIVE,
                    translateTransform * scaleTransform
                      * orientationTransform), layer.Get());

  // Draw lines radiating from center
  translateTransform = Matrix3x2F::Translation(outputBounds.Width / 2,
                                               outputBounds.Height / 2);

  for (float angle = 0; angle < 360; angle += 1)
  {
    Matrix3x2F rotationTransform = Matrix3x2F::Rotation(angle);
    context->SetTransform(rotationTransform * translateTransform *
                          orientationTransform);
    context->DrawLine(Point2F(0, 0),
                      Point2F(outputBounds.Width / 2, outputBounds.Height / 2),
                      m_whiteBrush.Get(), 2);
  }

  context->PopLayer();

  HRESULT hr = context->EndDraw();
  if (hr != D2DERR_RECREATE_TARGET)
  {
    DX::ThrowIfFailed(hr);
  }

  context->RestoreDrawingState(m_stateBlock.Get());
  m_needsRedraw = false;
  return true;
}
```

Once you've created an ID2D1PathGeometry object, you can use it for filling (in which case the result looks just like rendered text), drawing (which renders just the outlines), or clipping. **Figure 2** shows a Render method that scales and translates the path geometry to define a clipping region that encompasses the entire display area. Keep in mind the path geometry has both negative and positive coordinates. The (0, 0) origin of the path geometry corresponds to the baseline at the start of the glyph run.

> If you want to obtain character outline geometries in a Windows Store application, the Windows Runtime API won't help.

The Render method then draws a series of lines that radiate from the center of the screen, creating the image shown in **Figure 3**.

Deeper into Geometry Definitions

Generally speaking, a path geometry is a collection of figures, each of which is a collection of connected segments. These segments take the form of straight lines, quadratic and cubic Bézier curves, and arcs (which are curves on the circumference of an ellipse). A figure can be either closed, in which case the endpoint is connected to the start point, or open.

The GetGlyphRunOutline method writes the glyph outlines into an IDWriteGeometrySink, which is the same as an ID2D1SimplifiedGeometrySink. This in turn is the parent class to a regular ID2D1GeometrySink. Using ID2D1SimplifiedGeometrySink instead of ID2D1GeometrySink implies that the resultant path geometry contains figures consisting solely of straight lines and cubic Bézier curves—no quadratic Bézier curves and no arcs.

For font character outlines, these segments are always closed—that is, the endpoint of the figure connects to the start point. The path geometry created in the ClipToText program for the characters "CLIP" consists of five figures—one figure for each of the first three letters and two for the last letter to account for the inside of the upper part of the P.

Perhaps you'd like access to the actual lines and Bézier curves that make up the path geometry so you can manipulate them in weird and unusual ways. At first, this doesn't seem possible. Once an ID2D1PathGeometry object has been initialized with data, the object is immutable, and the interface provides no way to obtain the contents.

Figure 3 **The ClipToText Display**

Figure 4 **Structures for Saving the Contents of a Path Geometry**

```
struct PathSegmentData
{
  bool IsBezier;
  std::vector<D2D1_POINT_2F> Points;          // for IsBezier == false
  std::vector<D2D1_BEZIER_SEGMENT> Beziers;   // for IsBezier == true
};

struct PathFigureData
{
  D2D1_POINT_2F StartPoint;
  D2D1_FIGURE_BEGIN FigureBegin;
  D2D1_FIGURE_END FigureEnd;
  std::vector<PathSegmentData> Segments;
};

struct PathGeometryData
{
  D2D1_FILL_MODE FillMode;
  std::vector<PathFigureData> Figures;

  Microsoft::WRL::ComPtr<ID2D1PathGeometry>
                GeneratePathGeometry(ID2D1Factory * factory);
};
```

There is a solution, though: You can write your own class that implements the ID2D1SimplifiedGeometrySink interface, and pass an instance of that class to the GetGlyphRunOutline method. Your custom implementation of ID2D1SimplifiedGeometrySink must contain methods named BeginFigure, AddLines, AddBeziers and EndFigure (among a few others). In these methods you can save the entire path geometry in a tree of structures you can define.

This is what I did. The structures I defined for saving the contents of a path geometry are shown in **Figure 4**. These structures show how a path geometry is a collection of path figure objects, and each path figure is a collection of connected segments consisting of straight lines and cubic Bézier curves.

My implementation of the ID2D1SimplifiedGeometrySink is called InterrogableGeometrySink, so named because it contains a method that returns the resultant geometry as a PathGeometryData object. The most interesting parts of InterrogableGeometrySink are shown in **Figure 5**.

> Generally speaking, a path geometry is a collection of figures, each of which is a collection of connected segments.

Simply pass an instance of InterrogableGeometrySink to GetGlyphRunOutline to get the PathGeometryData object that describes the character outlines. PathGeometryData also contains a method named GeneratePathGeometry that uses the tree of figures and segments to create an ID2D1PathGeometry object you can then use for drawing, filling or clipping. The difference is that prior to calling GeneratePathGeometry, your program can modify the points that make up the line and Bézier segments. You can even add or remove segments or figures.

The InterrogableGeometrySink class and the supporting structures are part of a project named RealTextEditor; by "Real" I mean you can edit the text outlines instead of the text itself. When the program comes up, it displays the large characters "DX." Tap or click the screen to toggle edit mode. In edit mode, the characters are outlined and dots appear.

Green dots mark the beginnings and ends of line segments and Bézier segments. Red dots are Bézier control points. Control points are connected to corresponding endpoints with red lines. You can grab those dots with the mouse—they're a little too small for fingers—and drag them, distorting the text characters in weird ways, as **Figure 6** demonstrates.

RealTextEditor has no facility to save your custom character geometries, but you could certainly add one. The intent of this program isn't really to edit font characters, but to clearly illustrate how font characters are defined by a series of straight lines and Bézier curves connected into closed figures—in this case three figures, two for the inside and outside of the D and another for the X.

Algorithmic Manipulations

Once you have a path geometry definition in the form of structures such as PathGeometryData, PathFigureData, and PathSegmentData, you can also manipulate the individual points algorithmically, twisting and turning characters in whatever way you please, perhaps creating an image such as that shown in **Figure 7**.

Figure 5 **Most of the InterrogableGeometrySink Class**

```
void InterrogableGeometrySink::BeginFigure(D2D1_POINT_2F startPoint,
                                           D2D1_FIGURE_BEGIN figureBegin)
{
  m_pathFigureData.StartPoint = startPoint;
  m_pathFigureData.FigureBegin = figureBegin;
  m_pathFigureData.Segments.clear();
}

void InterrogableGeometrySink::AddLines(const D2D1_POINT_2F *points,
                                        UINT pointsCount)
{
  PathSegmentData polyLineSegment;
  polyLineSegment.IsBezier = false;
  polyLineSegment.Points.assign(points, points + pointsCount);
  m_pathFigureData.Segments.push_back(polyLineSegment);
}

void InterrogableGeometrySink::AddBeziers(const D2D1_BEZIER_SEGMENT *beziers,
                                          UINT beziersCount)
{
  PathSegmentData polyBezierSegment;
  polyBezierSegment.IsBezier = true;
  polyBezierSegment.Beziers.assign(beziers, beziers + beziersCount);
  m_pathFigureData.Segments.push_back(polyBezierSegment);
}

void InterrogableGeometrySink::EndFigure(D2D1_FIGURE_END figureEnd)
{
  m_pathFigureData.FigureEnd = figureEnd;
  m_pathGeometryData.Figures.push_back(m_pathFigureData);
}

HRESULT InterrogableGeometrySink::Close()
{
  // Assume that the class accessing the geometry sink knows what it's doing
  return S_OK;
}

// Method for this implementation
PathGeometryData InterrogableGeometrySink::GetPathGeometryData()
{
  return m_pathGeometryData;
}
```

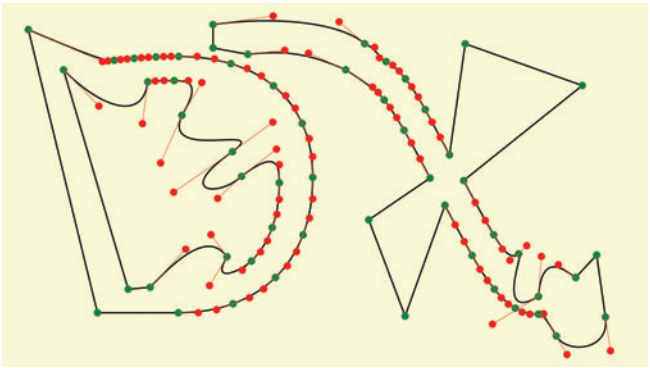


Figure 6 **Modified Character Outlines in RealTextEditor**

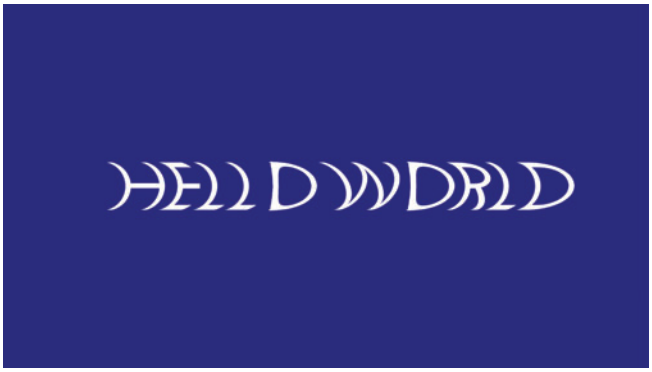


Figure 7 **The OscillatingText Program**

Well, not quite. The image shown in **Figure 7** is *not* possible using a PathGeometryData object generated from the InterrogableGeometrySink class I've just shown you. In many simple sans-serif fonts, the capital H consists of 12 points connected by straight lines. If you're dealing solely with those points, there's no way you can modify them so the straight lines of the H become curves.

However, you can solve that problem with an enhanced version of InterrogableGeometrySink called InterpolatableGeometrySink. Whenever this new class encounters a straight line in the AddLines method, it breaks that line into multiple smaller lines. (You can control this feature with a constructor argument.) The result is a completely malleable path geometry definition.

The OscillatingText program responsible for the image in **Figure 7** actually swings the interior of the characters back and forth, much like a hula dance. This algorithm is implemented in the Update method in the rendering class. Two copies of a PathGeometryData are retained: The source (identified as "src") describes the original text outline, and the destination ("dst") contains modified points based on the algorithm. The Update method concludes by calling GeneratePathGeometry on the destination structure, and that's what the program displays in its Render method.

Sometimes when algorithmically altering a path geometry, you might prefer working solely with lines rather than Bézier curves. You can do that. You can define an ID2D1PathGeometry object

Figure 8 **The TripleAnimatedOutline XAML File**

```
<Page
  x:Class="TripleAnimatedOutline.MainPage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:local="using:TripleAnimatedOutline"
  xmlns:dwritelib="using:SimpleDWriteLib">

  <Page.Resources>
    <dwritelib:TextGeometryGenerator x:Key="geometryGenerator"
                                     Text="Outline"
                                     FontFamily="Times New Roman"
                                     FontSize="192"
                                     FontStyle="Italic" />
  </Page.Resources>

  <Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="Auto" />
      <ColumnDefinition Width="*" />
    </Grid.ColumnDefinitions>

    <ListBox Grid.Column="0"
             ItemsSource="{Binding Source={StaticResource geometryGenerator},
                                   Path=FontFamilies}"

             SelectedItem="{Binding Source={StaticResource geometryGenerator},
                                    Path=FontFamily,
                                    Mode=TwoWay}" />
    <Path Name="path"
          Grid.Column="1"
          Data="{Binding Source={StaticResource geometryGenerator}, Path=Geometry}"
          Fill="LightGray"
          StrokeThickness="6"
          StrokeDashArray="0 2"
          StrokeDashCap="Round"
          HorizontalAlignment="Center"
          VerticalAlignment="Center">
      <Path.Stroke>
        <LinearGradientBrush StartPoint="0 0" EndPoint="1 0"
                             SpreadMethod="Reflect">
          <GradientStop Offset="0" Color="Red" />
          <GradientStop Offset="1" Color="Blue" />
          <LinearGradientBrush.RelativeTransform>
            <TranslateTransform x:Name="brushTransform" />
          </LinearGradientBrush.RelativeTransform>
        </LinearGradientBrush>
      </Path.Stroke>

      <Path.Projection>
        <PlaneProjection x:Name="projectionTransform" />
      </Path.Projection>
    </Path>
  </Grid>

  <Page.Triggers>
    <EventTrigger>
      <BeginStoryboard>
        <Storyboard>
          <DoubleAnimation Storyboard.TargetName="path"
                           Storyboard.TargetProperty="StrokeDashOffset"
                           EnableDependentAnimation="True"
                           From="0" To="2" Duration="0:0:1"
                           RepeatBehavior="Forever" />

          <DoubleAnimation Storyboard.TargetName="brushTransform"
                           Storyboard.TargetProperty="X"
                           EnableDependentAnimation="True"
                           From="0" To="2" Duration="0:0:3.1"
                           RepeatBehavior="Forever" />

          <DoubleAnimation Storyboard.TargetName="projectionTransform"
                           Storyboard.TargetProperty="RotationY"
                           EnableDependentAnimation="True"
                           From="0" To="360" Duration="0:0:6.7"
                           RepeatBehavior="Forever" />
        </Storyboard>
      </BeginStoryboard>
    </EventTrigger>
  </Page.Triggers>
</Page>
```

from a call to GetGlyphRunOutline, and then call Simplify on that ID2D1PathGeometry using the D2D1_GEOMETRY_SIMPLIFICATION_OPTION_LINES constant and an InterpolatableGeometrySink instance.

From DirectX to the Windows Runtime

If you're acquainted with the Windows Runtime API, the PathGeometryData, PathFigureData and PathSegmentData structures in **Figure 4** probably seem very familiar. The Windows::Xaml::UI::Media namespace contains similar classes named PathGeometry, PathFigure and PathSegment, from which PolyLineSegment and PolyBezierSegment derive. These are the classes you use to define a path geometry in the Windows Runtime, which you normally render using the Path element.

To demonstrate the interoperability of the SimpleDWriteLib component, the TripleAnimatedOutline application project is written in Visual Basic.

Of course, the similarity shouldn't be surprising. After all, the Windows Runtime is built on DirectX. What this similarity implies is that you can write a class that implements ID2D1SimplifiedGeometrySink to build a tree of PathGeometry, PathFigure, PolyLineSegment and PolyBezierSegment objects. The resultant PathGeometry object is directly usable by a Windows Runtime application and can be referenced in a XAML file. (You could also write an ID2D1SimplifiedGeometrySink implementation that generates a XAML representation of a PathGeometry and insert that into a XAML file in any XAML-based environment, such as Silverlight.)

The TripleAnimatedOutline solution demonstrates this technique. The solution contains a Windows Runtime Component project named SimpleDWriteLib that contains a public ref class named TextGeometryGenerator, which provides access to the system fonts and generates outline geometries based on these fonts. Because this ref class is part of a Windows Runtime Component, the public interface consists solely of Windows Runtime types. I made that public interface consist mostly of dependency properties so it could be used with bindings in a XAML file. The SimpleDWriteLib project also contains a private class named InteroperableGeometrySink that implements the ID2D1SimplifiedGeometrySink interface and constructs a Windows Runtime PathGeometry object.

Figure 9 **The RippleText Display**

You can then use this PathGeometry with a Path element. But watch out: When the Windows Runtime layout engine computes the size of a Path element for layout purposes, it only uses positive coordinates. To make the PathGeometry easier to use in a XAML file, TextGeometryGenerator defines a DWRITE_GLYPH_OFFSET that modifies coordinates based on the capHeight field of the font metrics structure. This serves to adjust the geometry coordinates to begin at the top of the font characters rather than at the origin, and to eliminate most negative coordinates.

To demonstrate the interoperability of the SimpleDWriteLib component, the TripleAnimatedOutline application project is written in Visual Basic. But don't worry: I didn't have to write any Visual Basic code. Everything I added to this project is in the MainPage.xaml file shown in **Figure 8**. The ListBox displays all the fonts on the user's system, and an outline geometry based on the selected font is animated in three ways:

- Dots travel around the characters;
- A gradient brush sweeps past the text;
- A projection transform spins it around the vertical axis.

A second program also uses SimpleDWriteLib. This is RippleText, a C# program that uses a CompositionTarget.Rendering event to perform an animation in code. Similar to OscillatingText, RippleText obtains two identical PathGeometry objects. It uses one as an immutable source and the other as a destination whose points are algorithmically transformed. The algorithm involves an animated sine curve that's applied to the vertical coordinates, resulting in distortions such as those shown in **Figure 9**.

Although the examples I've shown here are extreme in many ways, you certainly have the option to create subtler effects. I suspect that much of the WordArt feature in Microsoft Word is built around techniques involving the manipulation of character outlines, so that might provide some inspiration.

You can also integrate these techniques into more normal text-display code based on IDWriteTextLayout. This interface has a method named Draw that accepts an instance of a class that implements the IDWriteTextRenderer interface. That's a class you'll write yourself to get access to the DWRITE_GLYPH_RUN object that describes the text to be rendered. You can make changes to the glyph run and then render the modified version, or you can generate the character outline geometries at that point and modify the outlines prior to rendering.

Much of the power of DirectX lies in its flexibility and adaptability to different scenarios. ■

Charles Petzold *is a longtime contributor to* MSDN Magazine *and the author of "Programming Windows, 6th edition" (Microsoft Press, 2012), a book about writing applications for Windows 8. His Web site is charlespetzold.com.*

Thanks to the following technical expert for reviewing this article: *Jim Galasyn (Microsoft)*

DAVID S. PLATT

Original Sin?

When I want to impress a client, I take him to dinner at the Harvard Faculty Club. It's an excellent place, except for some of the low-life scum it has as members. And the club has pretty much forgiven me for that unfortunate incident with the chandelier years ago. Afterward I take him on a campus tour, which includes the Harvard Science Center, where I taught for years before the school moved me to a video studio.

The Science Center lobby contains the historic Harvard Mark I calculator (**Figure 1**), widely considered the world's first general-purpose programmable computation machine. It operated electro-mechanically rather than being purely electronic, with instructions fed in via paper tape. It was dedicated in August of 1944, about a year before the end of the Second World War.

The interpretive signs on the display are boring, even by academic standards, and downplay the fact that the very first thing this first computer did was calculate implosions as part of the Manhattan Project. It was only a year after World War II had ended that programmers working with the Mark I learned that the computer had helped drive the design of the atomic bomb dropped on Nagasaki.

It's hard to be indifferent to the atomic bombing that ended the Second World War. If you believe it to be the greatest war crime ever committed, then this machine should be preserved as a horrible reminder of technology servicing evil, as Germany to its credit has preserved its death camps and Japan to its shame has not. On the other hand, historian William Manchester, badly wounded as a Marine sergeant in the invasion of Okinawa, wrote: "You think of the lives which would have been lost in an invasion of Japan's home islands—a staggering number of American lives but millions more of Japanese—and you thank God for the atomic bomb" ("Goodbye, Darkness; A Memoir of the Pacific War," Little Brown, 1979). If you agree with Manchester, then the world contains few artifacts that have saved as many lives as this one (though Turing's Enigma-decrypting bombe [bit.ly/2KB0x] springs to mind), and its display should be a shrine.

The current display is neither. It's a somewhat interesting artifact to show my fellow geeks. We chuckle about how far we've come: this thing the size of a boxcar had less computing power than my toothbrush has today. Then my client and I go off and discuss how he can pay me more money. (I like that part.)

Harvard is missing an astounding educational opportunity here. I suspect that's because Harvard doesn't want to get into any controversy, having had enough when Larry Summers was president.

Sometimes I swing by the Science Center after teaching my evening class and just look at the Mark I and think about the bomb.

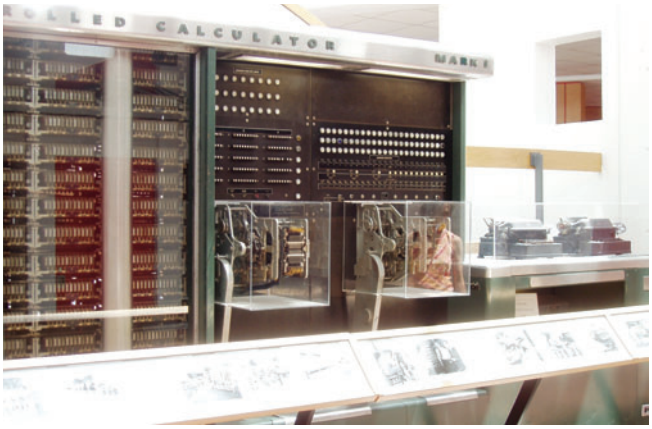


Figure 1 **The Harvard Mark I calculator was used in the design of the atomic bomb dropped on Nagasaki.**

Should we have dropped it? Should we have even developed it? I contemplate Harry Turtledove's alternate history novel "In the Presence of Mine Enemies" (Roc, 2004), where the Nazis got the bomb first. In the book, Turtledove describes a museum in Berlin that contains "behind thick leaded glass, the twisted radioactive remains of the Liberty Bell, excavated by expendable prisoners from the ruins of Philadelphia." Partly because of the men (and women—Grace Hopper was this machine's third programmer) who ran this beast, that timeline didn't happen. That suits me, and I sleep. And yet, and yet.

J. Robert Oppenheimer was the director of the Los Alamos lab that produced the first atomic bomb. He later wrote of those times: "In some sort of crude sense which no vulgarity, no humor, no overstatement can quite extinguish, the physicists have known sin; and this is a knowledge which they cannot lose."

Late at night, when the laughing undergraduates with their iPhones have drifted off in search of beer and sex, the lights are dim and the Science Center is quiet. I swear I hear the Mark I's relays clicking as it iterates through ghostly calculations, awaiting a HALT instruction that will never come from operators long dead. Am I hearing the faint echo of our industry's own original sin? ■

DAVID S. PLATT *teaches programming .NET at Harvard University Extension School and at companies all over the world. He's the author of 11 programming books, including "Why Software Sucks" (Addison-Wesley Professional, 2006) and "Introducing Microsoft .NET" (Microsoft Press, 2002). Microsoft named him a Software Legend in 2002. He wonders whether he should tape down two of his daughter's fingers so she learns how to count in octal. You can contact him at rollthunder.com.*

.NET TOOLS FOR DEV PROS

Whether you're building the most modern touch-enabled apps or maintaining and updating legacy applications, our flagship product, Studio Enterprise, helps to deliver rich, responsive, desktop and web apps on time and under budget.

NEW 2013 V3 THEME DESIGNER FOR WINFORMS

Visual Studio 2013 Launch Partner

ComponentOne® a division of GrapeCity®

DOWNLOAD YOUR FREE TRIAL

▸ componentone.com/se

CUTTING-EDGE MOBILE BUSINESS INTELLIGENCE SOLUTIONS

A NAME YOU CAN TRUST

Syncfusion has been in business for over a decade. With more than 7,000 customers, our frameworks drive some of the largest organizations in the world. Syncfusion is also trusted by over 100 Fortune 500 companies to power their key line-of-business applications.

Built on our powerful Orubase framework

One solution for iOS, Android, and Windows

Deploy to app stores with ease

Supports phone and tablet form factors

LEARN MORE TODAY.

syncfusion.com/BIdashboards

+1 888-9-DOTNET

Syncfusion®
Deliver innovation with ease®